eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Ingeniería de Computadores

Proyecto de Fin de Grado

Desarrollo de dos módulos de Gwyddion para análisis de defectos lineales en microscopía de efecto túnel.

Autor

Igor Cortés

Directores

Maxim Ilin e Iñigo Aldazabal

Codirector/Tutor

Andoni Arruti

informatika fakultatea
facultad de informática

23 de Junio de 2016

Agradecimientos

Lo primero de todo es dar las gracias a mis padres, Sebas y Gemma, al resto de mi familia y a mi novia, por enseñarme lo que sé hoy y por darme todo lo que tengo hoy. Siempre estaré agradecido.

También tengo que dar las gracias a todos los compañeros y profesores que he tenido durante estos años, por los buenos momentos y todo lo aprendido. En especial, a mi tutor del proyecto Andoni Arruti.

Por último, gracias a mis directores del Centro de Física de Materiales (CSIC-UPV/EHU), Maxim Ilin e Iñigo Aldazabal, por darme la oportunidad y confiar en mi para la realización de este proyecto. Siempre me habéis tratado amablemente y eso me ha hecho sentirme como en casa.

Gracias a todos.

Resumen

Durante la realización de este proyecto, se han diseñado y desarrollado 2 módulos para el programa multiplataforma de visualización y análisis de imágenes Gwyddion de software libre. Las imágenes que se han utilizado, se han obtenido mediante un microscopio de efecto túnel, que es un instrumento para tomar imágenes de superficies a nivel atómico. Los módulos que se han creado, son capaces de detectar las distancias entre los distintos elementos de la imagen y de etiquetar cada elemento con un valor único para su posterior análisis independientemente de los demás.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Introducción

Los métodos experimentales de la ciencia de superficies, permiten preparar y estudiar las superficies de los monocristales con precisión atómica. En concreto, las superficies de los cristales de bajo índice bien preparadas, consisten en terrazas constituidas por planos atómicos de tamaño característico de algunas micras, y separadas por escalones de la altura de un átomo. Un avance en las técnicas relacionadas con las superficies es el desarrollo de una tecnología que permita organizar las terrazas en estructuras periódicas con los parámetros controlados. Estas superficies son importantes tanto para la investigación básica (para estudiar el crecimiento y las propiedades de las capas ultrafinas de diferentes materiales), como para la industria de la catálisis.

El método de la caracterización microscópica de las superficies cristalinas es la microscopía de efecto túnel (Scanning Tunneling Microscopy - STM). A través de esta técnica se generan imágenes con la resolución atómica que proporcionan información sobre las alturas y las distancias entre los elementos de la superficie. El análisis estadístico de las imágenes de STM permite deducir los parámetros característicos del conjunto de escalones y terrazas: la distribución de las distancias entre los escalones y la función de correlación a lo largo del escalón.

Mediante las técnicas para la detección de bordes, se pretende identificar los escalones en las imágenes STM para después poder analizarlas. En la siguiente figura, se puede observar una imagen STM, una imagen con los escalones detectados mediante la detección de bordes, y por último, una imágen en la que se ha realizado un tratamiento de los datos para conseguir valores de 0 y 1 en los píxeles.

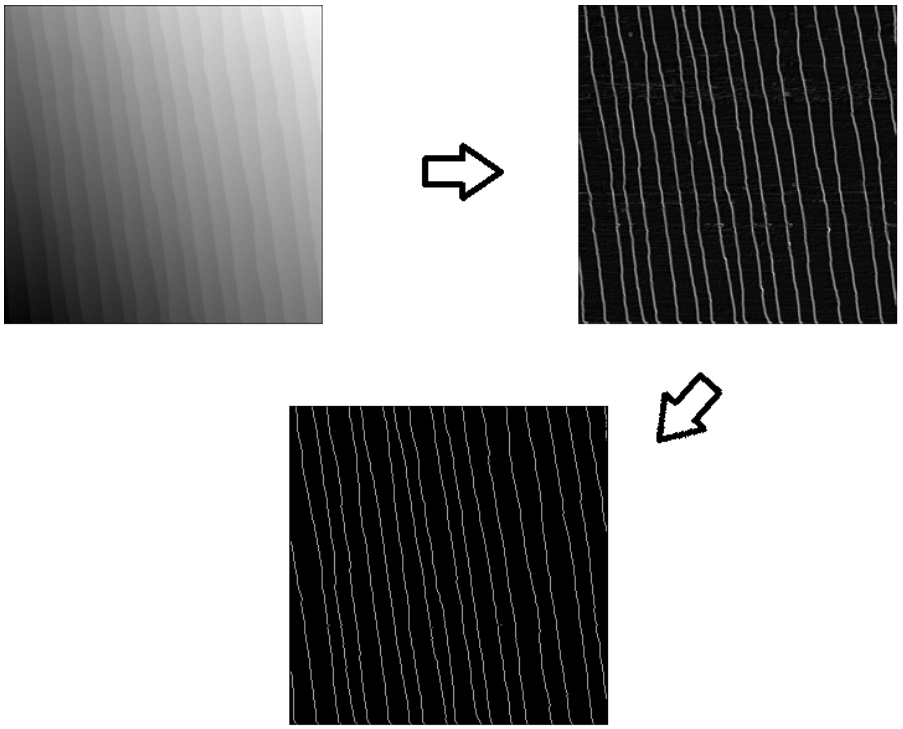

Figura 1.1: Imagen STM arriba a la izquierda, imagen obtenida mediante la detección de bordes arriba a la derecha y una imagen con valores de 0 y 1 abajo.

La detección de bordes es una técnica fundamental para el análisis digital en procesamiento de imágenes (2D y 3D) y visión por computador, concretamente en la detección y extracción de características, con la finalidad de encontrar los puntos en los que la intensidad del brillo cambia repentinamente. Cuando cambia el brillo en las imágenes, generalmente corresponde a uno de los siguientes casos:

- Discontinuidades en la profundidad.
- Discontinuidades en la orientación de las superficies.
- Cambios en las propiedades del material.
- Variaciones en la iluminación de la escena.

Las técnicas para la detección de bordes, es algo que se utiliza en muchos casos hoy en día, como en los coches autónomos para la detección de las líneas en la carretera, en aplicaciones para el reconocimiento en imágenes de rostros u objetos, en el análisis de imágenes médicas,... o como en este caso, en el análisis de imágenes microscópicas para estudiar la superficie.

1.1. Motivación

El proyecto consiste en el desarrollo de dos módulos de análisis estadístico para el programa de visualización y análisis de imágenes Gwyddion. Este proyecto se ha desarrollado para optimizar el tiempo de un trabajador del Centro de Física de Materiales (CSIC-UPV/EHU). Por tanto, las necesidades de este proyecto han sido descritas por el Dr Maxim Ilin, que a partir de ahora, se le mencionará como el cliente. Este proyecto ha estado dirigido por tres personas: los directores del CFM Dr Iñigo Aldazabal y Dr Maxim Ilin, y el codirector/tutor de la facultad Dr Andoni Arruti. Con todos ellos se ha mantenido contacto durante la realización del proyecto.

El cliente tiene programas de tratamiento de imágenes de STM, pero estos programas no tienen las herramientas de análisis estadístico especializados para trabajo con imagenes de superficies escalonados. Esta situación le obligaba a utilizar diferentes programas: una para el tratamiento de imágenes y otra para el análisis estadístico. Gracias a la arquitectura abierta de Gwyddion, se puede unir todas las funciones en un programa, lo que hace las cosas más convenientes y rápidas. De esta forma, se mejora la productividad teniendo en cuenta la importancia de las tareas.

1.2. Objetivos

El objetivo de este proyecto, consiste en el desarrollo de dos módulos para realizar un estudio estadístico en el programa de visualización y análisis de imágenes Gwyddion.

El primer módulo, tiene que ser capaz de distinguir los escalones atómicos para realizar un estudio sobre la distribución de las distancias entre los escalones.

El segundo módulo, tiene que ser capaz de etiquetar los distintos escalones con identificadores únicos, para después, poder tratar cada escalón independientemente de los demás. El tratamiento que se va a realizar a cada escalón, será un ajuste lineal, para encontrar la orientación promedia del escalón y luego calcular la desviación de esta orientación mediante una correlación a lo largo del escalón.

2. CAPÍTULO

Entorno de trabajo.

En este capítulo, se explicará cual ha sido el entorno de trabajo en el que se ha realizado el proyecto, tanto hardware como software.

2.1. Hardware

Para este proyecto, el principal dispositivo que se ha utilizado, ha sido la estación de trabajo en el Centro de Física de Materiales (CSIC-UPV/EHU). Un ordenador DELL con sistema operativo de 64 bits.

2.2. Software

El sistema operativo de la estación de trabajo, es Windows 7 Enterprise de 64 bits y se han utilizado los siguiente programas y librerías.

- Gwyddion: Es un programa modular y multiplataforma de software libre para la visualización y análisis de datos.
- GTK: Biblioteca que contiene los objetos y funciones para crear la interfaz de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, pestañas, ...

- gwy: Gwyddion proporciona un módulo de extensión de Python. De esta forma, podemos utilizar sus funciones y clases en un script.
- gwyutils: Contiene varias funciones de utilidad. Las más utilizadas serán data_field_data_as_array(field) y data_field_set_data(field, data). La primera función, crea una matriz numpy utilizando una imagen. La segunda, establece los datos de una matriz numpy a una imagen.
- numpy: Es el paquete fundamental para la computación científica con Python. Añade soporte para grandes arrays y matrices multidimensionales, además de una gran colección de funciones matemáticas para operar en estos arrays.
- scipy: Es otro paquete con herramientas y algoritmos matemáticos para python. Este paquete solo se utilizará para realizar un ajuste lineal.
- matplotlib: Es una biblioteca que permite generar gráficos a partir de las listas o matrices en el lenguaje de programación Python y su extensión matemática numpy.

2.2.1. Gwyddion

Como hemos mencionado antes, Gwyddion es un programa modular, desarrollado bajo licencia GPL, orientado al análisis de mapas de altura obtenidos mediante técnicas de microscopía de barrido. El programa posee un gran número de funciones y módulos para el procesado de las imágenes, y permite añadir nuevos análisis mediante una API escrita tanto en C como en Python. Los principales desarrolladores de este software son David Nečas (Yeti) y Petr Klapetek, quienes trabajan conjuntamente con varios desarrolladores de todo el mundo.

Para la instalación de este programa, unicamente nos tenemos que descargar el paquete de su página web www.gwyddion.net

2.2.2. Pygwy

Python permite automatizar todo el procesamiento de datos que queramos realizar. Para poder crear un script en Python e integrarlo en el programa Gwyddion, es necesario utilizar el módulo de Pygwy. Este módulo viene con el archivo de instalación del programa de Gwyddion, pero es necesario instalar por separado Python y PyGTK.

En este caso, se han utilizado las versiones que recomiendan tanto de Python como de PyGTK. Para instalar Python, se ha utilizado la versión 2.7 y para la instalación de PyGTK se ha utilizado la 2.24.0, pero es necesario instalar 2 paquetes más para evitar problemas, ya que depende de los siguientes paquetes: PyGObject-2.28.3 y PyCairo-1.8.10.

Todos los módulos externos al programa que se escriban en Python, deberán estar alojados dentro del subdirectorio pygwy, localizado en Documents and Settings\gwyddion\pygwy.

2.2.3. Python

La razón de utilizar Python para crear los módulos, se describen en la siguiente lista:

- Facilidad a la hora de desarrollar el código y una curva de aprendizaje más accesible que otros lenguajes.
- Contiene varias librerías para cálculos estadísticos y visualización gráfica como numpy, scipy y matplotlib.
- Los módulo que se desarrollan en Python para el programa Gwyddion, no son necesarios compilarlos cada vez que se hagan cambios o se quieran añadir en otro ordenador, solo hay que copiarlos dentro del directorio de módulos Pygwy.

Existía la posibilidad de realizar los módulos en C, ya que todos los módulos de este programa vienen escritos en dicho lenguaje, pero como su programación iba a resultar más complicada y los ficheros necesitan ser compilados cada vez que se realizasen cambios, se ha decidido hacerlo en Python. Más adelante, en el capítulo 7, se explicarán las ventajas de haber escrito los módulos en C.

2.2.4. Anaconda

Anaconda es una distribución multiplataforma de software libre de los lenguajes de programación Python y R, desarrollada por Continuum Analytics. Contiene una colección muy grande de librerías y paquetes, para el procesamiento de datos a gran escala y computación científica. Es una distribución, que permite trabajar con distintas versiones de Python, como la 2.7 o 3.3, con tan solo indicarlo. Anaconda posee su propio gestor de paquetes llamdo Conda, con el que se podrá instalar, crear y actualizar paquetes adicionales.

La gran ventaja de utilizar esta plataforma, es que contiene la aplicación Jupyter-Notebook, la cual permite prototipar de una forma rápida y sencilla. Tiene soporte para más de 40 lenguajes de programación, como Python, R, Julia y Scala. Los ficheros que se crean, son fácilmente compartidos con otras personas, como puede ser vía email, Dropbox o Github. Utilizar esta herramienta, permite visualizar los datos con los que se está trabajando en formato de imágenes, vídeo, Latex y JavaScript. Además, los widgets interactivos se pueden utilizar para manipular y visualizar datos en tiempo real.

3. CAPÍTULO

Módulo para la detección de escalones atómicos e histograma de distancias.

Para crear este módulo y poder integrarlo en el programa Gwyddion, existen 2 formas: escribirlo en C o en Python. El cliente, ha decidido que se haga en Python, por las razones que se describen en el capítulo 2.

3.1. Introducción

Gwyddion incluye unos algoritmos para la detección de bordes, pero el resultado de estos, no es el que desea exactamente el cliente.

Como se ha dicho, incluye varios algoritmos para la detección de bordes. Se ha trabajado con todos ellos y finalmente se han elegido 2 de ellos, inclination y RMS. Estos 2 algoritmos son con los que se ha obtenido mejores resultados para poder cumplir el objetivo del cliente.

Para crear este módulo, es necesario añadir una serie de librerías. Para poder interactuar con el programa Gwyddion, es necesario importar lo siguiente:

```
1 import  gwy, gwyutils
```

Las librerías de gwy y gwyutils, son las que permiten interactuar los módulos con el

programa gwyddion, ofreciendo funciones como la de visualizar la imagen, conseguir la matriz de la imagen, mostrar los nuevos cambios de la imagen,...

Además, se han añadido 2 librerías más: una librería para cálculos científicos en Python (numpy) y otra para realizar trazados 2D en Python (matplotlib).

```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

3.2. Detección de escalones

Como se ha mencionado antes, en este proyecto no se ha implementado una detección de bordes, sino que se han utilizado, principalmente, 2 de los algoritmos que aporta el programa para la detección de bordes. Estos, son los algoritmos que te permite utilizar Gwyddion.

- Canny
- Harris Corner
- Hough Lines
- Inclination
- Laplacia of Gaussian
- Local Nonlinearity
- Prewitt
- RMS
- RMS Edge
- Sobel
- Step
- Zero Crossing

Los algoritmos con los que se ha trabajado, han sido Inclination principalmente y RMS. La decisión de por qué se han elegido estos algoritmos, solo se basa en la obtención de mejores resultados. Se han utilizado todos ellos con las distintas imágenes con las que se ha trabajado, pero al final, se han elegido esos dos. A continuación, se hará una breve descripción sobre el funcionamiento de dichos algoritmos sin entrar en detalles.

3.2.1. Técnica RMS

Visualiza las áreas con alta variación de valor local. Se calcula y se muestra el cuadrado medio de la raíz de las desviaciones del valor medio de una vecindad circular de radio 2,5 píxeles centrada alrededor de cada muestra.

3.2.2. Técnica Inclination

Visualiza el ángulo θ de la inclinación del plano local.

3.3. Filtro para la detección de escalones

Las imágenes con las que se van a trabajar, son unas imágenes en blanco y negro como las siguientes:

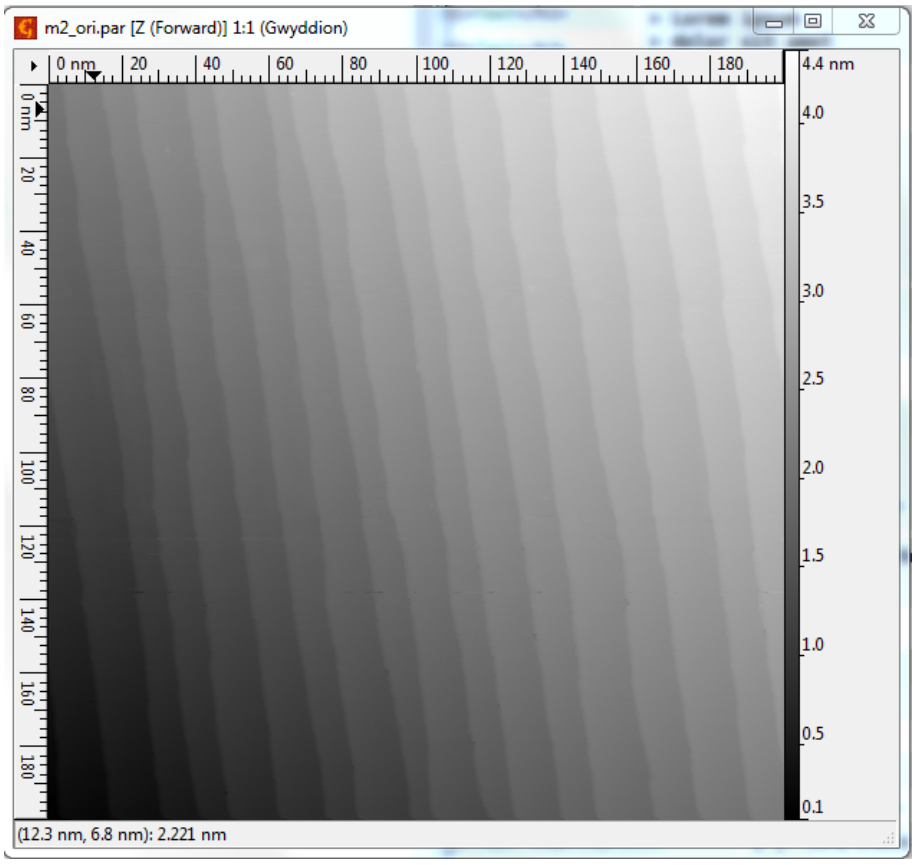

Figura 3.1: Imagen escaneada a una superficie de cristal con unos pocos escalones.

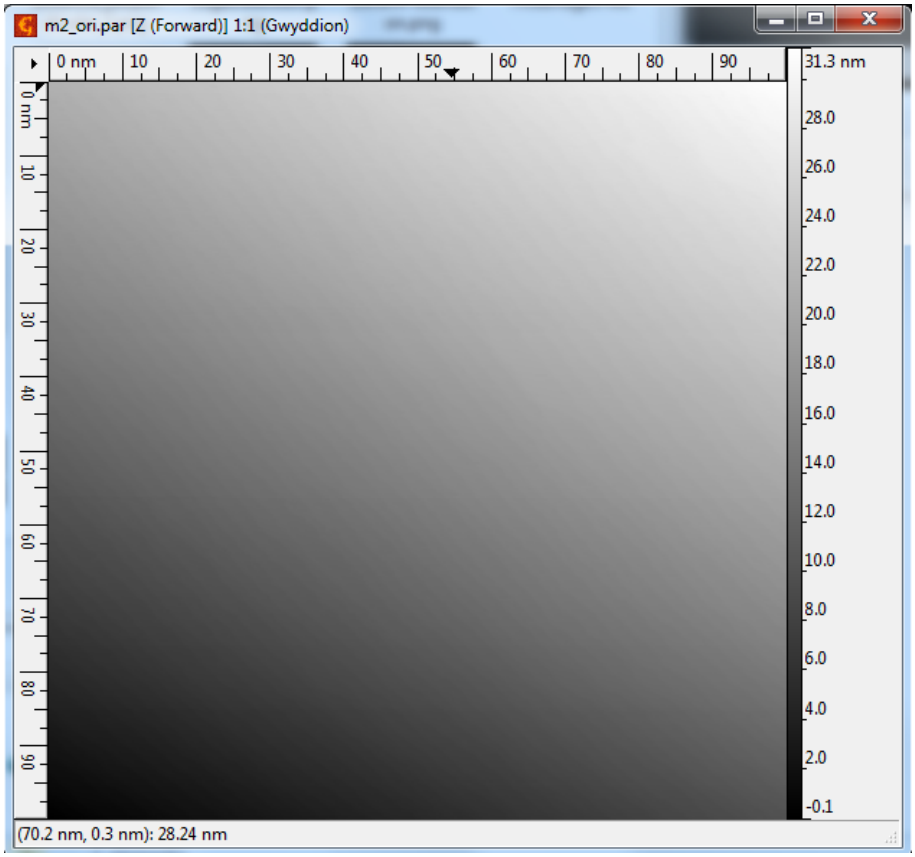

Figura 3.2: Imagen escaneada a una superficie de cristal con muchos más escalones.

En la figura 3.1 se pueden apreciar los escalones. Para detectar los escalones de las figuras 3.1 y 3.2, utilizaremos uno de los algoritmos para la detección de bordes que nos aporta Gwyddion. Estos algoritmos se pueden encontrar en el menú Data Process $\longrightarrow$ Presentation $\longrightarrow$ Edge Detection. Aquí, se utilizará el algoritmo inclination como se ha mencionado antes y nos devolverá la figura 3.3 como resultado de la figura 3.1 y la figura 3.4 como resultado de la figura 3.2.

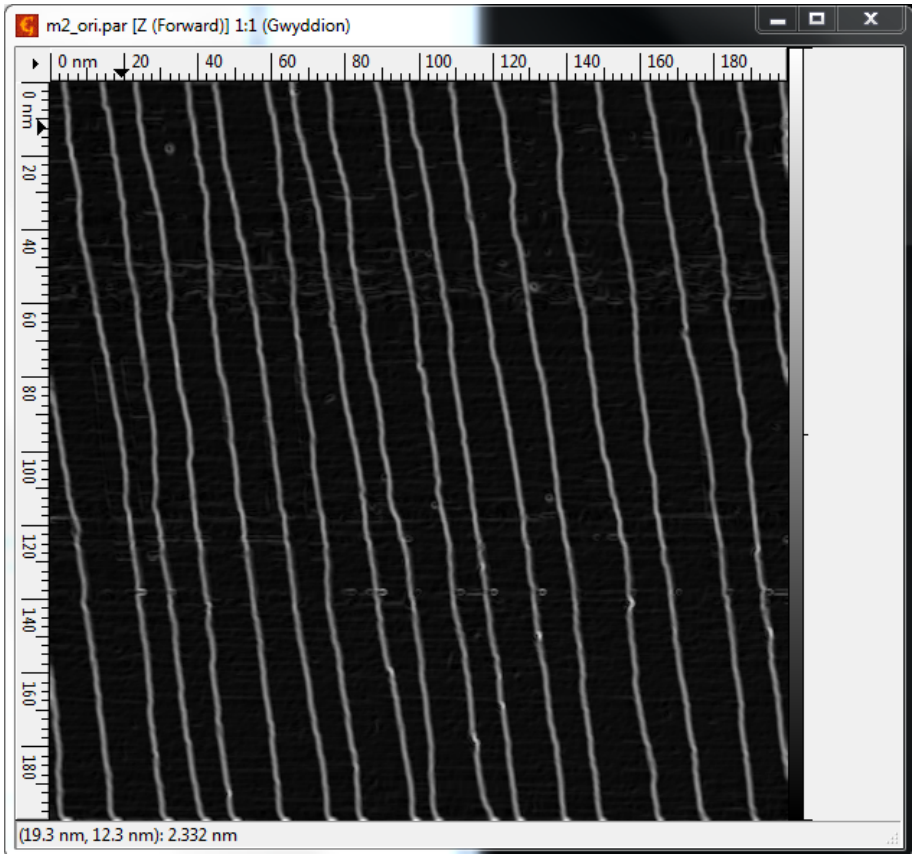

Figura 3.3: Imagen procesada con el algoritmo inclination.

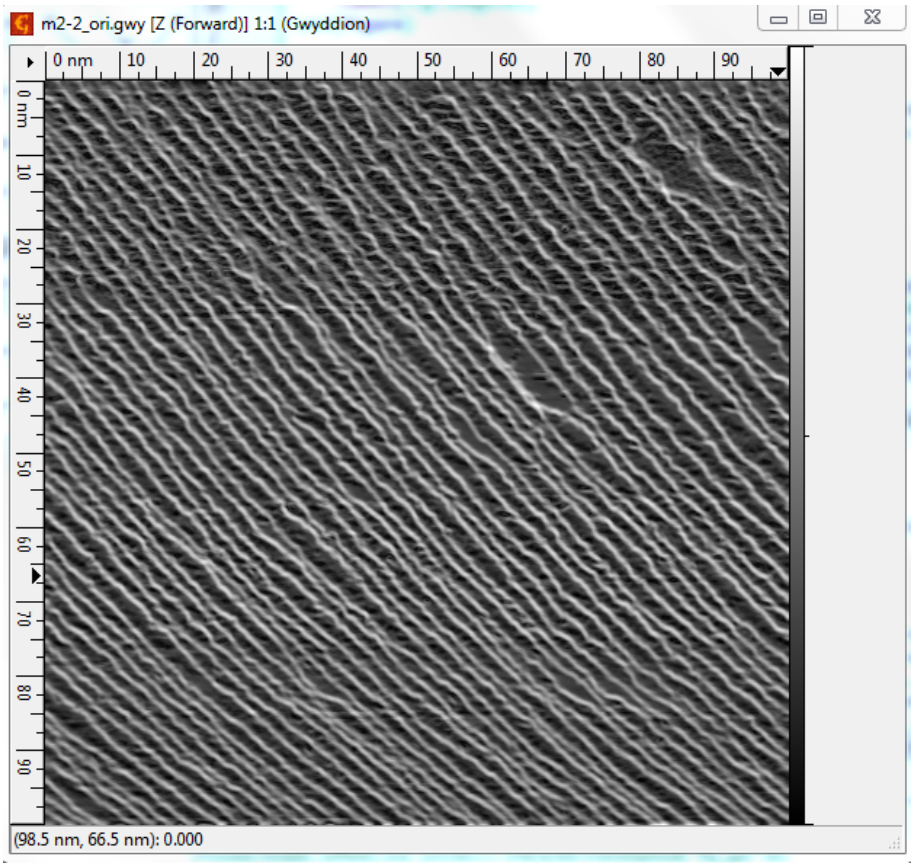

Figura 3.4: Imagen procesada con el algoritmo inclination.

Una vez obtenidas las imágenes con los escalones detectados, es imprescindible realizar un filtro. La matriz de la figura 3.3 como de la figura 3.4, contienen valores entre 0 y 1. Lo que se ha hecho, ha sido tratar la imagen línea por línea, detectando los picos. Una forma de detectar los picos en datos, es usar la propiedad de que un pico debe ser mayor que sus vecinos inmediatos.

En este caso, se han necesitado 2 parámetros a la hora de detectar los picos, los cuales son la altura y la distancia. En altura, detectamos los picos que son mayores que la altura de pico mínima que se indica. En distancia, se detectarán los picos que están al menos separados por la distancia de pico mínima. Las figuras 3.5, 3.6 y 3.7 muestran una representación gráfica de alguna de las lineas de la figura 3.3.

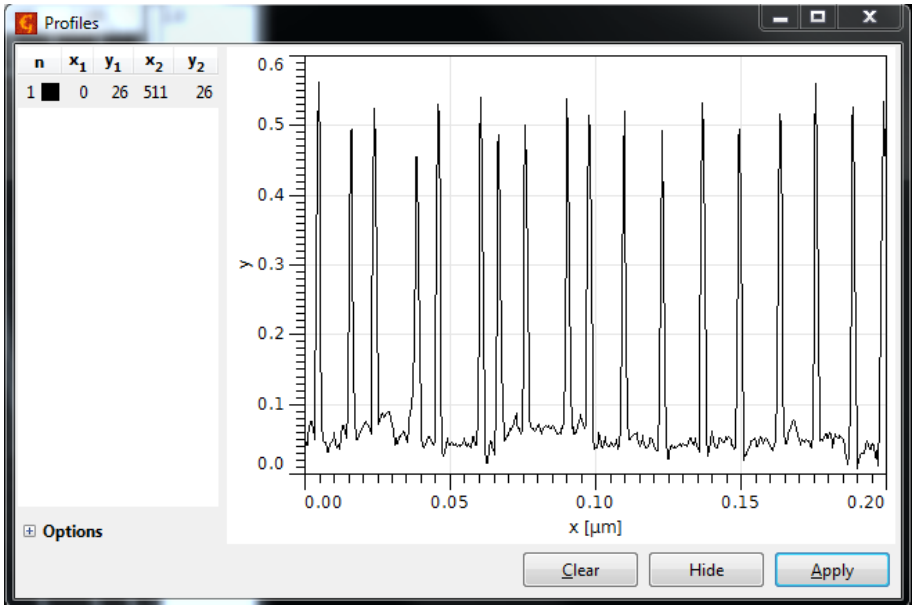

Figura 3.5: Representación gráfica de la línea 26 de la figura 3.3.

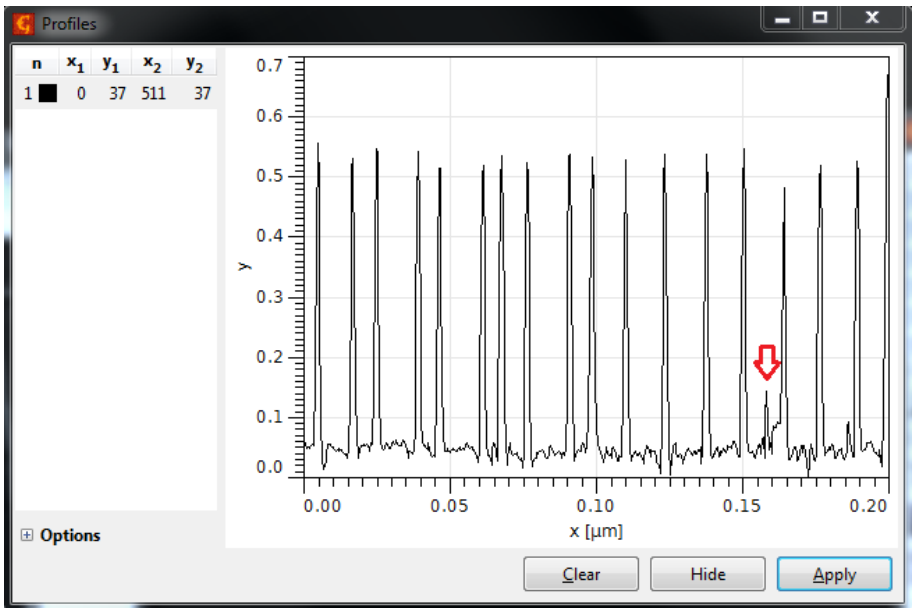

Figura 3.6: Representación grafica de la línea 37 de la figura 3.3, se puede apreciar que un pico no tiene la altura del resto.

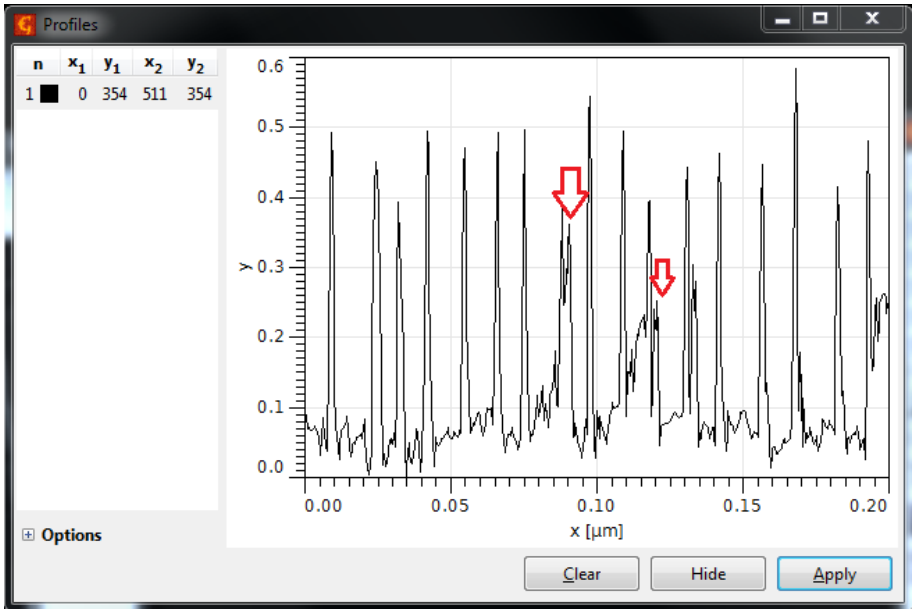

Figura 3.7: Representación gráfica de la línea 354 de la figura 3.3, se puede apreciar que unos picos no tienen la suficiente distancia entre ellos como el resto.

Para detectar los picos en función de los 2 parámetros que se han indicado más arriba, se ha utilizado la función detect_peaks de **Marcos Duarte**. Una vez obtenidos los picos detectados, se creará otra matriz con el mismo tamaño de la imagen, la cual solo tendrá valores de 0 y 1. Los valores de 1 representarán los píxeles donde se encuentra el escalón y los valores de 0 representarán el resto de píxeles.

En el siguiente código, se puede observar que se utiliza la función data_field_data_as_array de la librería gwyutils. Como su nombre indica, esta función consigue la matriz de la imagen y se guarda en una variable. Es imprescindible realizar la transposición de la matriz, ya que cuando el programa carga en memoria la matriz de la imagen, internamente realiza una transposición.

Una vez se tiene la matriz de la imagen correctamente, se ajustarán los parámetros a las características de los picos de la imagen (mph = altura mínima del pico, mpd = distancia mínima entre picos) y el algoritmo detect_peaks recorrerá la matriz línea a línea obteniendo los picos. Obtenidos los picos, se marcarán con un 1 dichas posiciones en una matriz repleta de 0. De esta forma, tenemos una nueva matriz en la que hemos eliminado ruidos y se obtiene una imagen más limpia.

```
data.array = gwyutils.data_field_data_as_array(data.data_field)
data.array = np.transpose(data.array)

tam = data.array.shape
zerosa = np.zeros(tam)

for x in range (0,zerosa.shape[0]):
   ind = detect_peaks(data.array[x], mph=float(mph), mpd=int(mpd),show=False)

   for y in range (0, ind.size):
      zerosa[x][ind[y]] = 1
```

Por último, solo faltaría realizar de nuevo la transpuesta de la matriz, ya que al utilizar la función data_field_set_data de la librería gwyutils, también realiza una transposición internamente.

```
data.array = zerosa
data.array = np.transpose(data.array)
data.another_field.data_changed()
gwyutils.data_field_set_data(data.another_field, data.array)
```

Una vez se han detectado todos los picos y actualizamos los datos de la imagen, la imagen que obtenemos serían las siguientes figuras: 3.3 y 3.4:

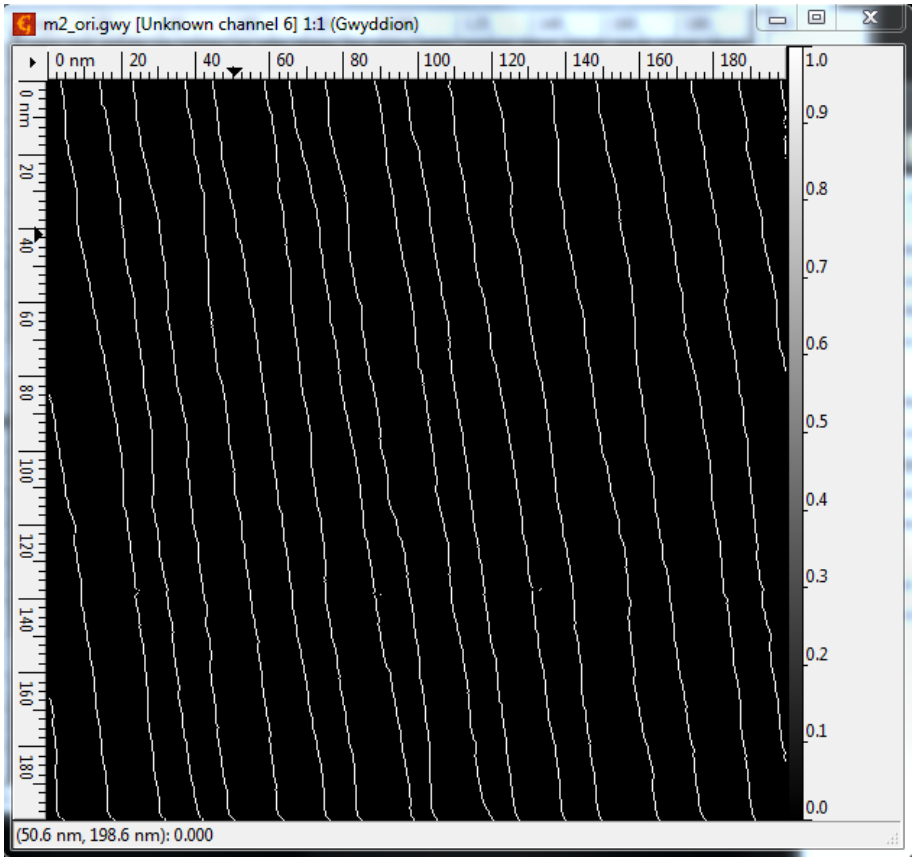

Figura 3.8: Imagen filtrada de la figura 3.3, donde los escalones tienen el valor 1 y el resto de la imagen 0.

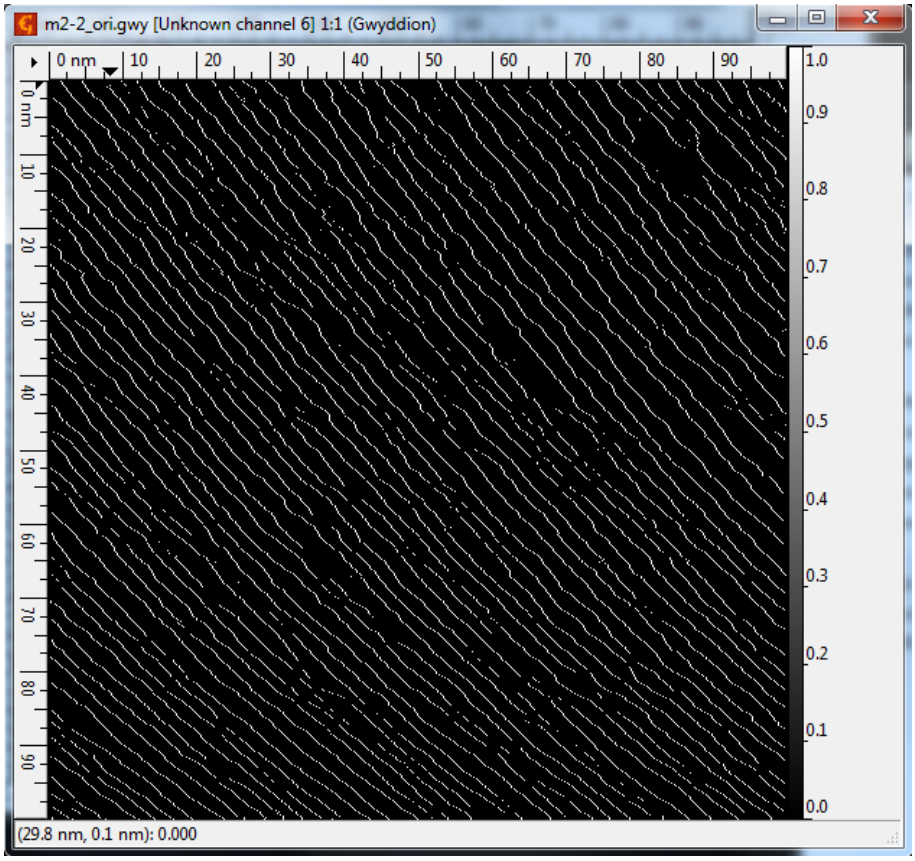

Figura 3.9: Imagen filtrada de la figura 3.4, donde los escalones tienen el valor 1 y el resto de la imagen 0.

Tabla 3.1: Representación de la matriz de una imagen filtrada con el algoritmo detect_peaks de **Marcos Duarte**

0	1	0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	0	1	0	0

3.4. Cálculo de las distancias e histograma

Uno de los intereses del cliente, es poder calcular la distancia perpendicular línea a línea entre 2 escalones, para después, poder verlas en un histograma e incluso poder guardarlas en un fichero para su estudio. Estas distancias se han calculado en nanómetros.

Para conseguir la distancia perpendicular entre 2 escalones (línea azul de la figura 3.10), hemos calculado la distancia horizontal entre 2 picos (línea roja de la figura 3.10). Como se puede observar en la figura 3.10, tenemos un triangulo recto, por tanto, el cálculo de la línea azul es trivial.

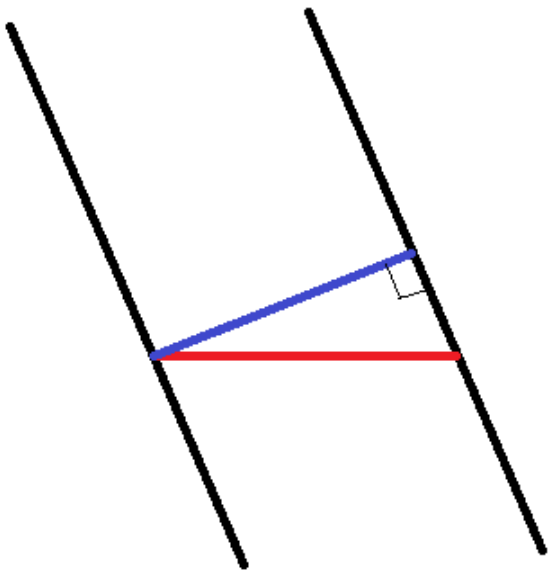

Figura 3.10: Representación para el calculo perpendicular entre 2 líneas.

Como se puede observar en la figura 3.11, la distancia que se necesita calcular, sería el

adyacente, y la distancia que conocemos es la hipotenusa. Por tanto, para calcular la distancia del adyacente, se ha tenido que multiplicar la hipotenusa por el coseno del ángulo. El ángulo, lo podemos obtener de una forma un tanto rudimentaria. Se ha utilizado una herramienta de rotar la imagen del propio programa de Gwyddion. Esta herramienta se encuentra en el menú Data Process ⟶ Basic Operations ⟶ Rotate by angle.

```
1 Funcion coseno: cos(α) = Adyacente / Hipotenusa
2 Adyacente = cos(α) * Hipotenusa
```

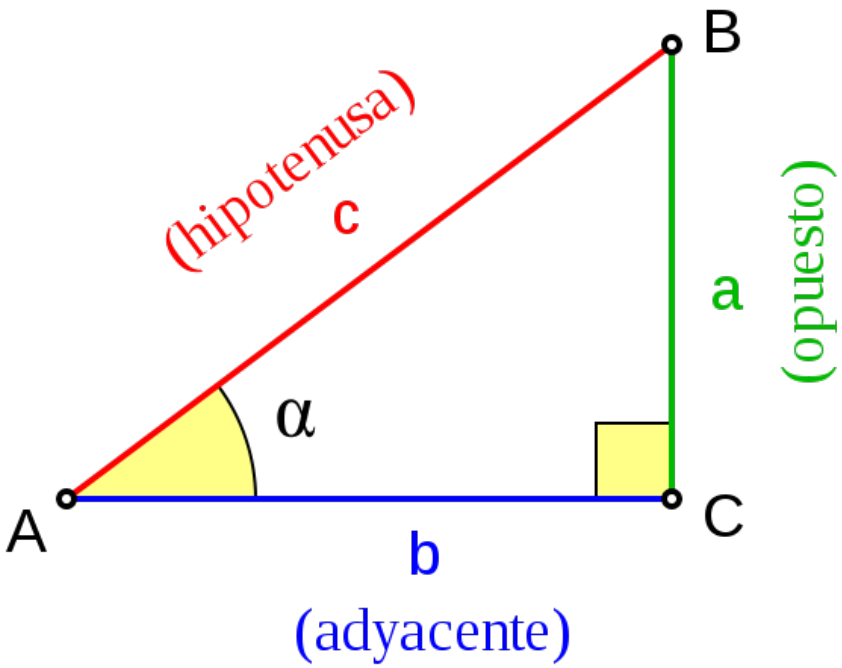

Figura 3.11: Representación de un triangulo recto nombrando sus lados.

De esta forma, si recorremos línea a línea la matriz de la imagen, conseguiremos todas las distancias en perpendicular entre los escalones. Existe la posibilidad de cometer algunos errores, ya que los escalones detectados de algunas imágenes no son líneas continuas. Por lo tanto, la distancia que obtendríamos no sería la de 2 escalones contiguos como se puede apreciar en la figura 3.12.

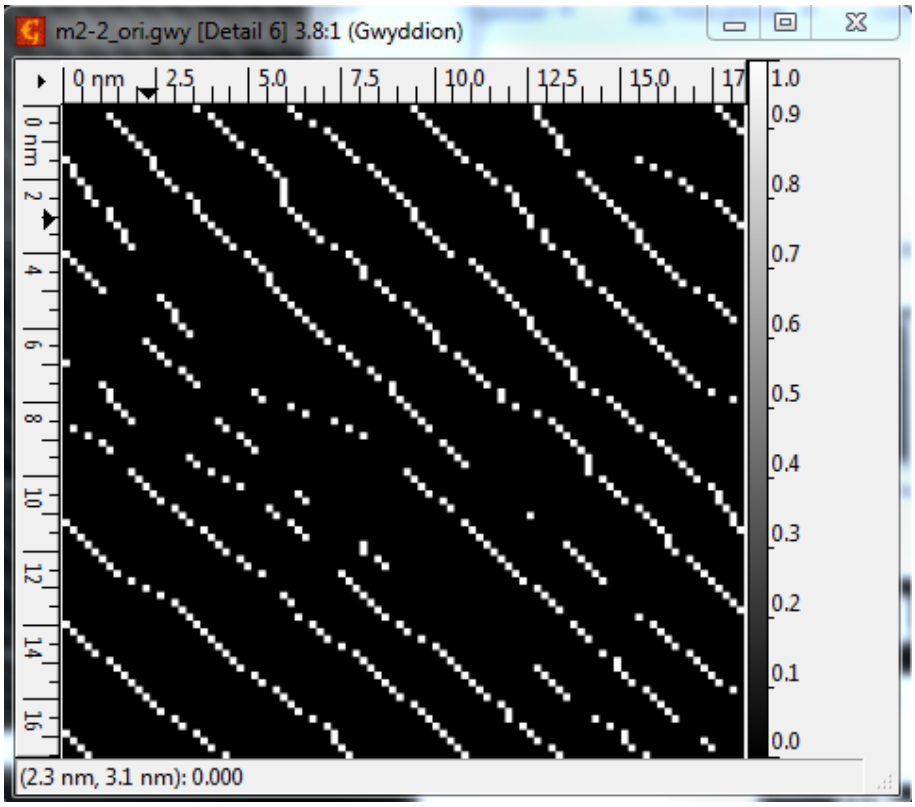

Figura 3.12: Recorte de una imagen en la que los escalones no son lineas continuas.

Con las distancias calculadas, solo queda visualizarlas en un histograma. Para ello, se ha escrito el siguiente código:

```
histlista = []
data.histarray = np.transpose(data.array)
data.grados = float(data.entry_grados.get_text())
data.hbins = float(data.entry_bins.get_text())
for x in range(0,data.histarray.shape[0]):
   a = False
   b = False
   y1 = 0
   y2 = 0
   for y in range(0,data.histarray.shape[1]):
      if data.histarray[x][y] == 1 and a == False:
         a = True
         y1 = y
      elif data.histarray[x][y] == 1 and a == True and b == False:
         b = True
         y2 = y
         histlista.append((float(y2-y1))*(data.data_field.get_xreal()/data.data_fiel.
             get_xres())**(10**9)*np.cos((2*np.pi/360)*data.grados))
         y1 = y2
         b = False
```

Como se puede observar en el código, se ha escrito un algoritmo que detecta la distancia entre un escalón y el siguiente, contando el número de píxeles. Esta distancia en píxeles, la convertimos a una distancia real. Para calcular el tamaño real en horizontal de un píxel, se han utilizando las funciones get_xreal() y get_xres(), que devuelven la distancia real de la imagen en metros y el número de píxeles que tiene la imagen en horizontal. Por tanto, el tamaño real de un píxel, sería la división del tamaño real de la imagen entre el número total de píxeles. Luego, se ha realizado un cambio de unidad de metros a nanómetros, multiplicando la distancia por 10^9 para obtener la distancia en nanómetros. Esto último a petición del cliente.

```
bins = np.arange(np.floor(min(histlista)),np.ceil(max(histlista)),data.hbins)
plt.hist(histlista, bins, (min(histlista),max(histlista)))
plt.grid(True)
plt.title('Distances histogram')
plt.xlabel('Distances in nanometers')
plt.ylabel('Count')
plt.show()
```

Por último, multiplicamos la distancia por el coseno del ángulo de inclinación. El ángulo lo introducimos a mano, utilizando una herramienta de rotación que hemos mencionado antes. El intervalo para el histograma también será introducido a mano por el usuario.

Una vez hemos calculado las distancias, podemos visualizarlas en un histograma para su análisis, como se puede ver en la figura 3.13.

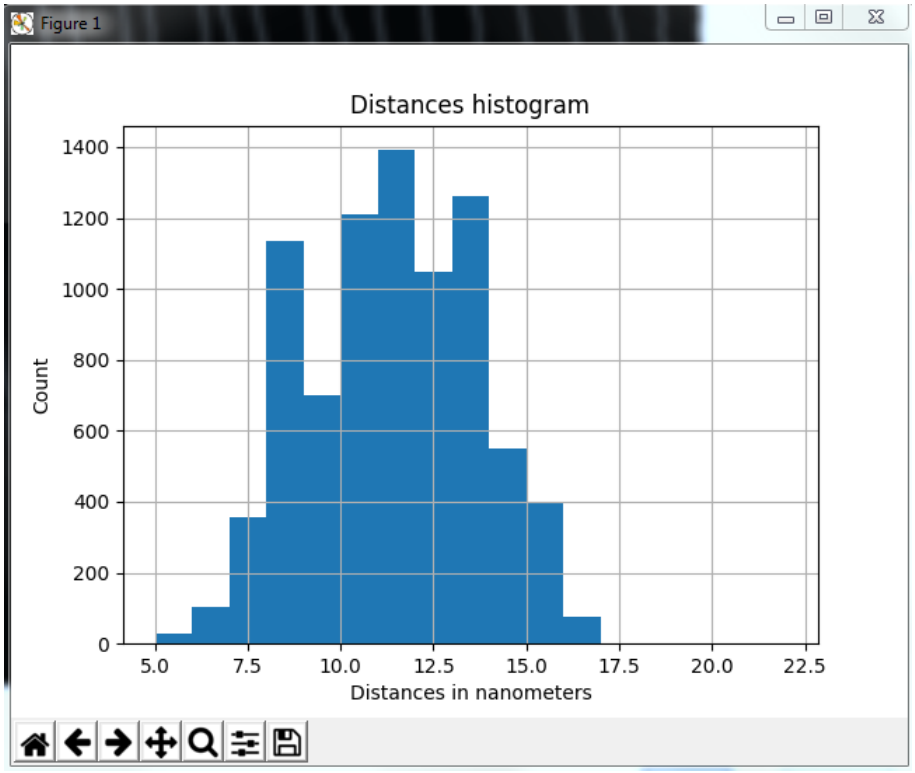

Figura 3.13: El histograma de la imagen 3.8 introduciendo un ángulo de inclinación de 8 grados y un intervalo de 1.

4. CAPÍTULO

Módulo para el etiquetado y análisis estadístico de escalones.

Este módulo, al igual que el anterior, se ha escrito en Python para facilitar su uso a las personas.

4.1. Introducción.

El segundo objetivo del cliente, ha sido la posibilidad de etiquetar los escalones y realizar un análisis estadístico de los escalones. El cliente estaba interesado en poder seleccionar un escalón, independientemente de los demás, para poder calcular la desviación que sufren a la hora de escanear la superficie.

En este módulo, se ha añadido un nuevo módulo de la librería scipy, concretamente stats.

```
1 from scipy import stats
```

Este módulo contiene un gran número de distribuciones de probabilidad, así como una creciente biblioteca de funciones estadísticas. Se ha añadido para poder realizar un ajuste lineal más adelante.

4.2. Etiquetado de los escalones.

El etiquetado de los escalones es algo esencial para que se pueda tratar cada uno de ellos independientemente de los demás. Para ello, se ha creado un algoritmo que recorre línea por línea la matriz de la imagen. En el momento que se detecta un escalón, se ha identificado con un índice único, y se continuará marcando en su dirección hasta el final. Esto es posible porque se conoce el dato de la pendiente.

Para averiguar el siguiente punto del escalón, se ha utilizado un método que nos indicará si 2 puntos pertenecen a la misma línea recta.

$$(y - y_1) = m * (x - x_1)$$

(x1,y1) es un punto específico, y se quiere localizar el siguiente punto (x,y). Como la imagen se recorre en una dirección ascendente, el siguiente punto (x,y) siempre estará en la siguiente línea, por tanto la coordenada "y" sería la siguiente:

$$y = y1 + 1.$$

Para poder calcular la pendiente "m", se utiliza el ángulo de inclinación de estos escalones, que es posible obtenerlo con una herramienta para rotar la imagen del programa. Una vez que se obtiene el ángulo, para averiguar la pendiente, solo se necesita multiplicar el ángulo por la tangente.

$$m = tg(ángulo).$$

Por último, solo quedaría averiguar en qué columna x estaría el siguiente punto (x,y). Para ello, se despeja la x de la anterior fórmula y quedaría lo siguiente:

$$x = \frac{y - y_1}{tg(angulo)} + x_1$$

Al no ser una línea recta, no siempre va a estar el siguiente punto en dicha coordenada "x", en ocasiones, es posible encontrarse la coordenada "x" que nos interesa desviada unas posiciones a la derecha o hacia la izquierda. Esto es debido a que en ocasiones, los escalones pueden entrelazarse entre ellos, no han superado el filtro y no han sido marcados, al escanear la superficie ha ocurrido algún fallo,... como se puede observar en las figuras 4.1 y 4.2.

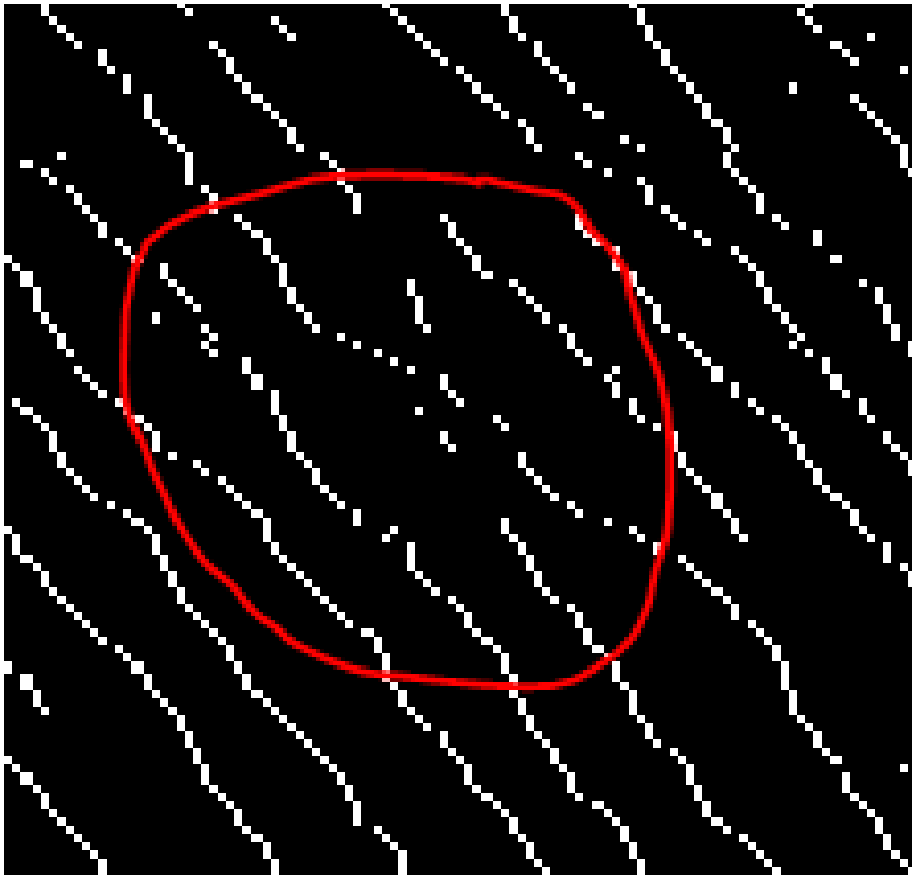

Figura 4.1: Escalones de la figura 3.9 entrelazándose

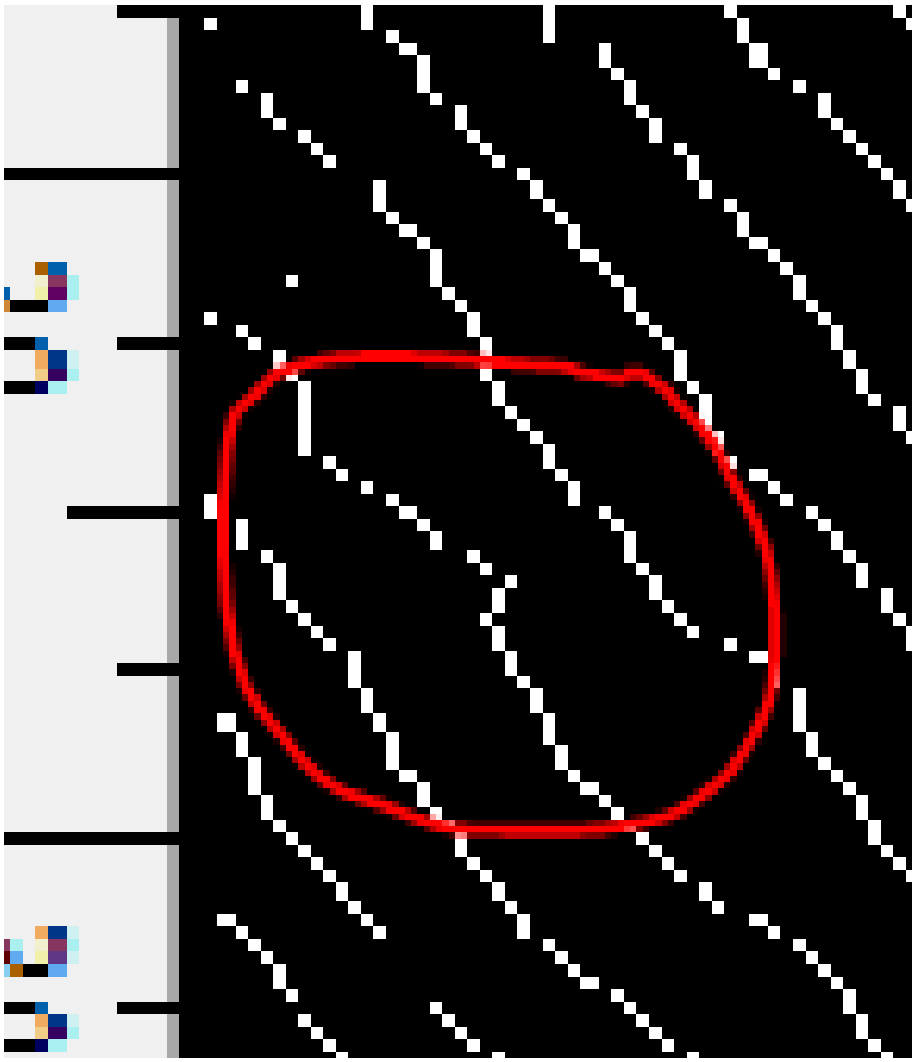

Figura 4.2: Escalon de la figura 3.9 que no sigue la pendiente

Para los escalones que se entrelazan no se puede hacer nada, pero para los escalones que no siguen la pendiente, se ha tenido que introducir una nueva variable, con la que se le indica cuantos puntos a la izquierda y a la derecha se tienen que comprobar, para poder encontrar el siguiente punto. En el caso de encontrar la nueva coordenada "x" desviada, se seguiría identificando la línea desde ese mismo punto. En el caso de no encontrar ningún punto, se saltaría a la siguiente línea, ya que como se ha mostrado antes en la figura 3.12, los escalones no son siempre líneas continuas.

En el siguiente código, se muestra como se ha conseguido etiquetar los escalones con un índice único.

```
...
for row in range(0,aux.shape[0]):
   for column in range(0,aux.shape[1]):
      if aux[row][column] == 1 and data.labelMatrix[row][column] == 0:
         lineCountFinal += 1
         data.labelMatrix =labelingLine(aux,row,column,angulo,data.labelMatrix,lineCountFinal,
         scanWidthLeft,scanWidthRight)
...
```

En el código se puede observar que se analiza la imagen línea por línea, buscando un escalón que no esté etiquetado. En el momento que se encuentra un píxel con esas características, se llama a la función labelingLine(...), que es la encargada de marcar ese escalón hasta el final.

```
def labelingLine(aux,row,column,angulo,labelMatrix,lineCountFinal,scanWidthLeft,scanWidthRight):

   currentRow=row
   currentColumn=column

   labelMatrix[currentRow][currentColumn]=lineCountFinal

   nextRow = currentRow + 1
   nextColumn=int((((nextRow)-currentRow)/(np.tan((2*np.pi/360)*(90-angulo))))+currentColumn)
```

Lo primero que realiza, es marcar la coordenada con el identificador único, después, se calculan las nuevas coordenadas del siguiente punto del escalón. Para ello, se ha utilizado el método explicado anteriormente para buscar las nuevas coordenadas (x,y).

```
while(nextRow < aux.shape[0] and nextColumn < aux.shape[1]):
   currentRow = nextRow
   currentColumn = nextColumn

```

```
    if aux[currentRow][currentColumn] == 1 and labelMatrix[currentRow][currentColumn] == 0:
        labelMatrix[currentRow][currentColumn] = lineCountFinal
        nextRow += 1
        nextColumn=int((((nextRow)-currentRow)/(np.tan((2*np.pi/360)*(90-angulo))))+currentColumn)
    else:
        if currentColumn - scanWidthLeft < 0:
            rango_inicio = currentColumn
        else:
            rango_inicio = scanWidthLeft

        if currentColumn + scanWidthRight > aux.shape[1]-1:
            rango_final = aux.shape[1]-1-currentColumn
        else:
            rango_final = scanWidthRight

        existePunto=False
        for columnPoint in range(currentColumn-rango_inicio,currentColumn+rango_final):
            if aux[currentRow][columnPoint] == 1 and labelMatrix[currentRow][columnPoint] == 0:
                existePunto = True
                break
        if existePunto == True:
            labelMatrix[currentRow][columnPoint] = lineCountFinal
            nextRow += 1
            #columnPoint
            nextColumn=int((((nextRow)-currentRow)/(np.tan((2*np.pi/360)*(90-angulo))))+columnPoint)
        else:
            nextRow += 1
            nextColumn=int((((nextRow)-currentRow)/(np.tan((2*np.pi/360)*(90-angulo))))+
      currentColumn)

    return labelMatrix
```

Una vez se tienen las nuevas coordenadas, la primera comprobación que se realiza, es si esas coordenadas están dentro de las dimensiones de la imagen. En caso de estar dentro, se comprobará si ese píxel tiene el valor 1, confirmando que pertenece a un escalón, y si no ha sido etiquetado anteriormente. Si todo es correcto, se etiquetará el punto con el mismo valor único y se buscarán las siguientes coordenadas del escalón.

En el caso de que la siguiente coordenada no tuviese el valor de 1, se realizará una búsqueda de varios píxeles, definida por el usuario, a la izquierda y a la derecha. Primero, se realizará una comprobación de los píxeles que se van a mirar, de esta forma, en el caso de salirse de las dimensiones de la imagen, se modificarán dichas variables y se verificará que no se sale de las dimensiones de la imagen. Si se encontrase un píxel con el valor 1 y que no esté etiquetado, se ha decidido que el escalón continúa desde ese punto como se ha explicado anteriormente.

Por último, cuando se detecta que se ha llegado al final de las dimensiones de la imagen,

se sale de esta función, devolviendo la matriz con los escalones etiquetados, aumentar el valor del identificador único, buscar el siguiente escalón y volver a realizar el mismo proceso.

En la figura 4.3, se muestra un diagrama de flujo del algoritmo labelingLine para verlo mejor.

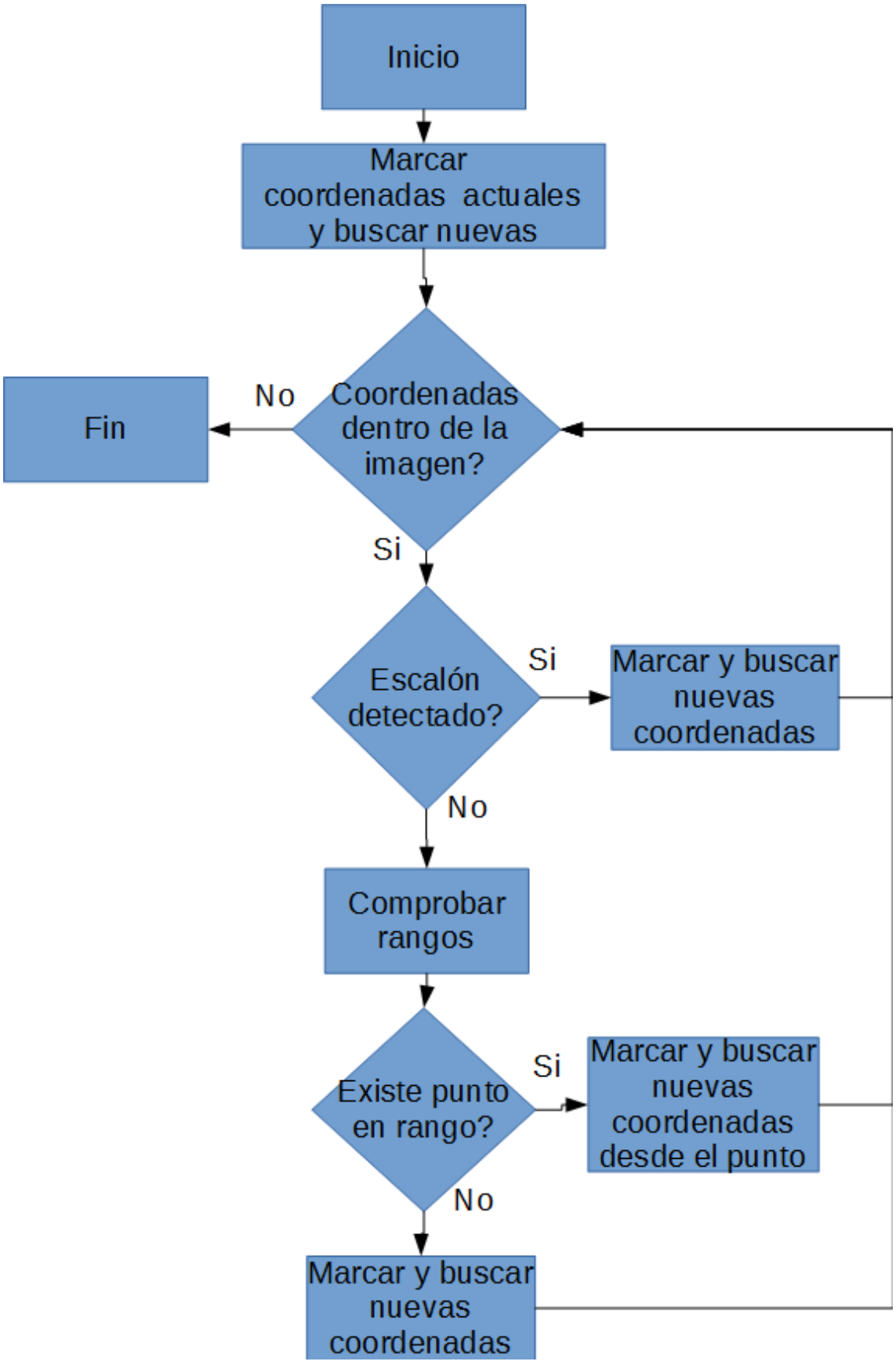

Figura 4.3: Diagrama de flujo del algoritmo labelingLine()

En la figura 4.4, se puede ver una representación de los escalones identificados con un valor único.

$$\begin{matrix}
0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 \\
0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 \\
0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0
\end{matrix}$$

Figura 4.4: Representación de los escalones identificados

4.3. Seleción de líneas

Una vez se han etiquetado los escalones, como se puede observar en la figura 4.4, se ha implementado una función para poder seleccionar cada uno de ellos y visualizarlo. De esta forma, se puede comprobar que escalones han sido identificados correctamente y cuales no, ya que por problemas mencionados anteriormente, puede darse el caso de que algunos escalones se identifiquen erróneamente.

Para seleccionar un escalón y poder resaltarlo de los demás, se ha utilizado una escala de grises, donde el escalón seleccionado se mostrará con un color blanco y el resto de los escalones, se mostrarán grises. El resto de la imagen será negra. Esto lo realiza la función selecting_one_line(), donde el usuario introducirá que escalón quiere visualizar.

```
1  def selecting_one_line(data):
2  ...
3  data.selected_line =int(data.spinn_line.get_value())
4
5  for row in range(0,tam[0]):
6     for column in range(0, tam[1]):
7        if data.labelMatrix[row][column] == 0:
8           data.ret[row][column]=0
9
10       elif data.labelMatrix[row][column] != data.selected_line:
11          data.ret[row][column]=0.4
12
13       else:
```

```
14          data.ret[row][column]=1
15  ...
```

En el código, se puede observar que se recoge la línea que se quiere seleccionar y se recorre toda la matriz con los escalones etiquetados. Si el píxel tiene el valor 0, se dejará con el mismo valor, ya que indica que es el fondo de la imagen. Si el valor del píxel, no es la línea que se desea seleccionar, se le dará un valor de 0.4, indicando de esta forma que es un escalón, pero no el deseado por el usuario, y se verá en gris. El resto de píxeles, tienen que contener el valor del escalón que se quiere seleccionar y se les dará el valor de 1 para que se vean blancos.

En las figuras 4.5, 4.6, 4.7, 4.8 y 4.9 se puede observar que podemos elegir el escalón con el que queremos trabajar.

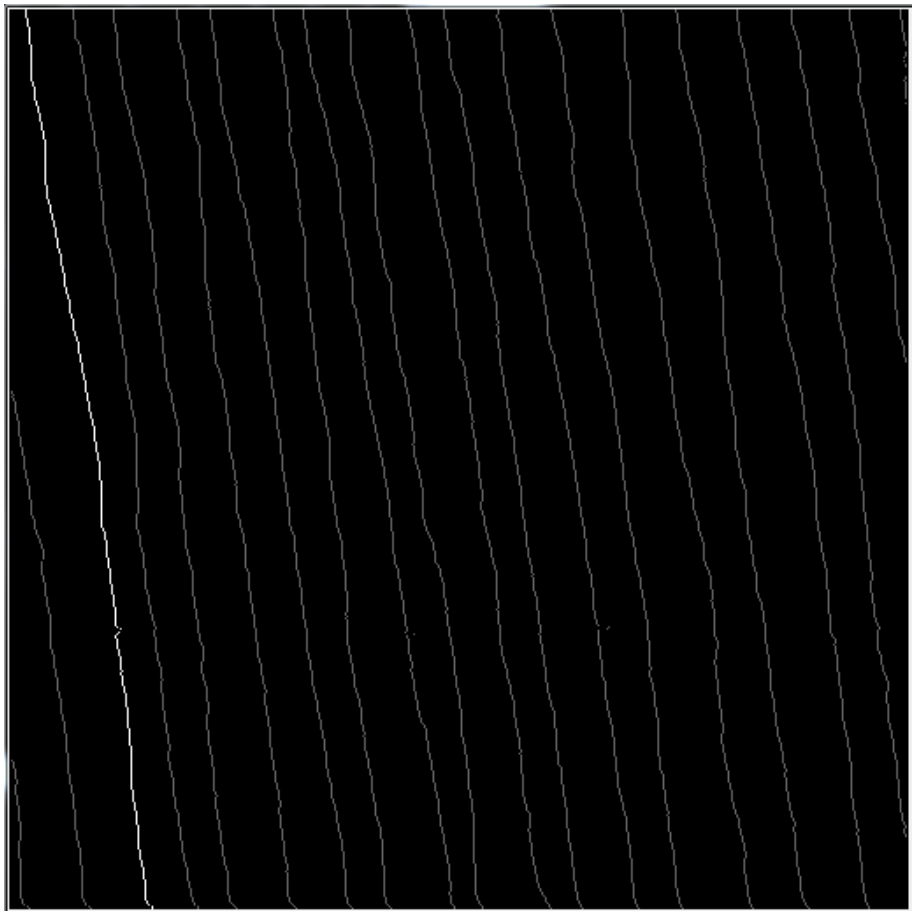

Figura 4.5: Seleccionamos la primera línea de la figura 3.8.

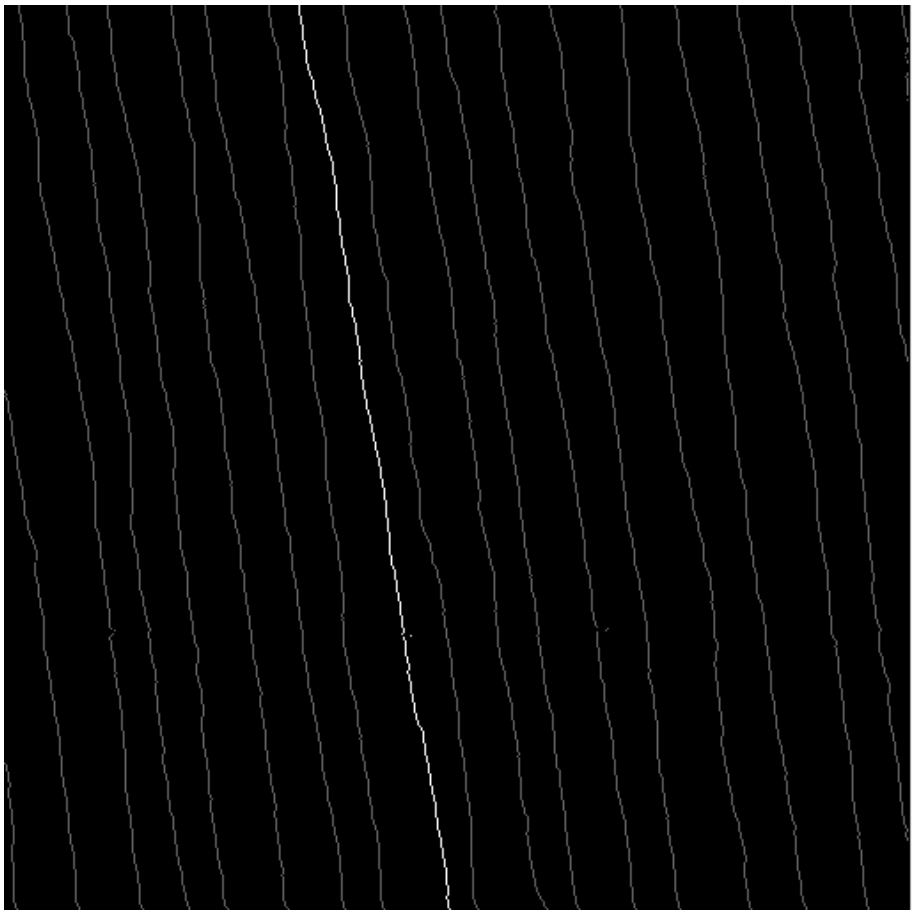

Figura 4.6: Seleccionamos la sexta línea de la figura 3.8.

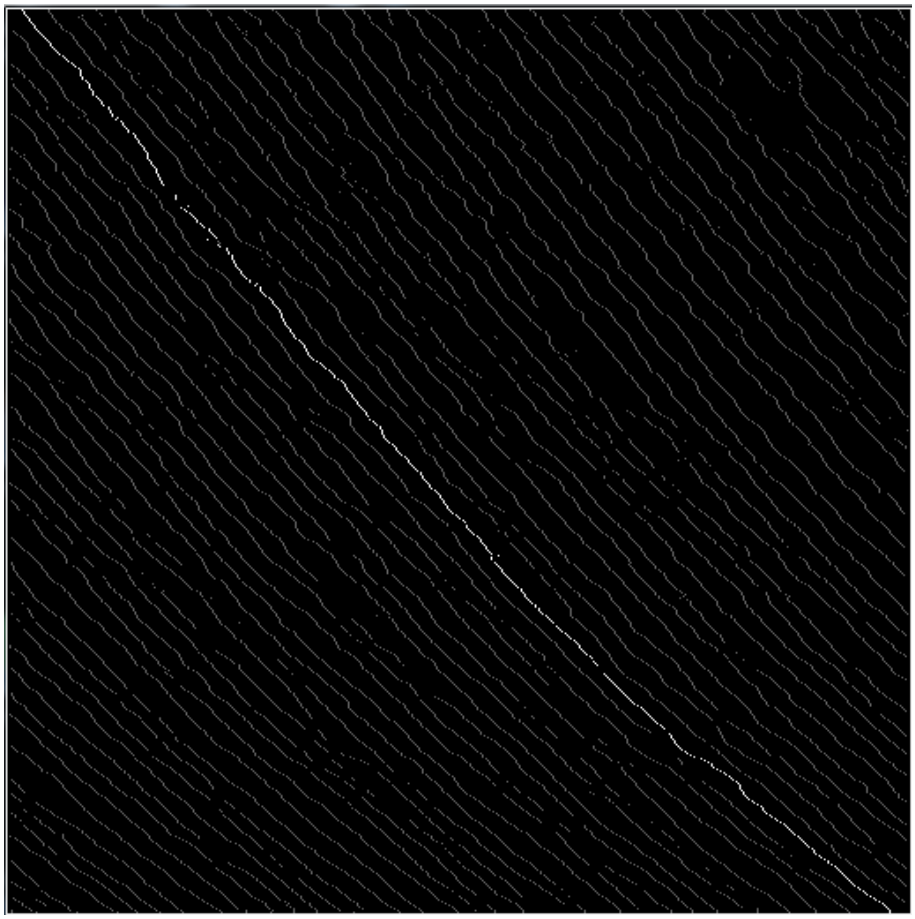

Figura 4.7: Seleccionamos la sexta línea de la figura 3.9.

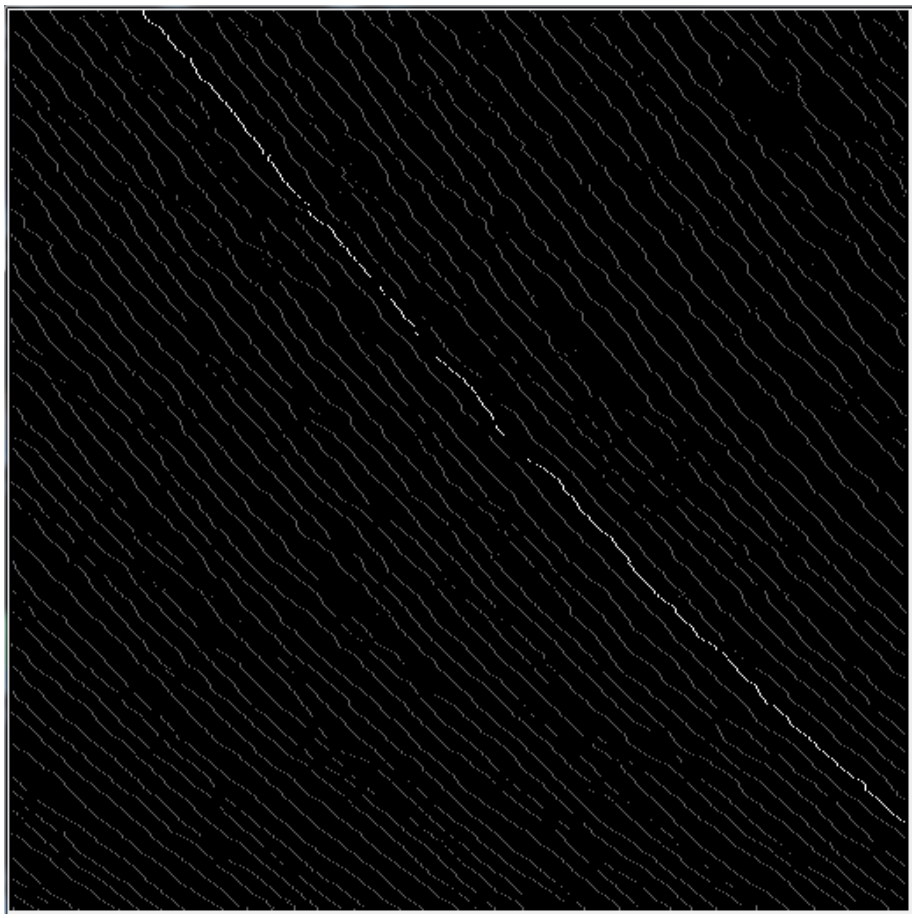

Figura 4.8: Seleccionamos la quinta línea de la figura 3.9.

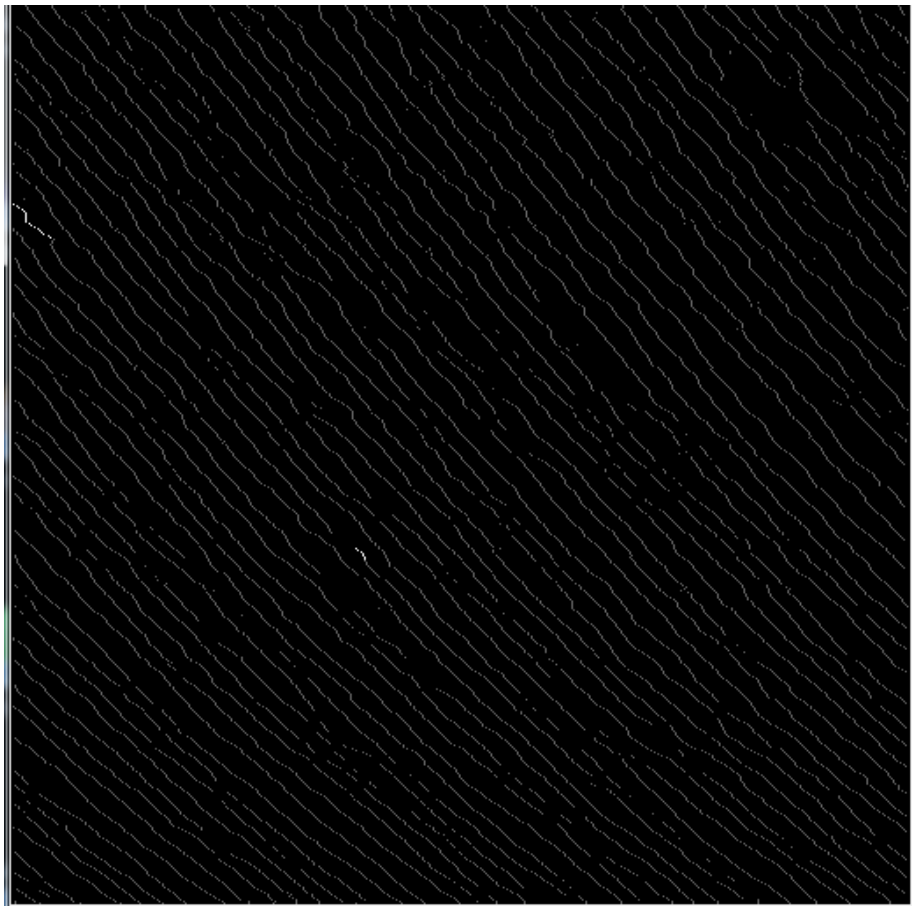

Figura 4.9: Seleccionamos la sexagésima segunda línea de la figura 3.9.

En el caso de las figuras 4.5, 4.6, 4.7 y 4.8, se puede observar que las líneas de los escalones han sido bien etiquetadas. En cambio, en la figura 4.9, se puede apreciar que la línea del escalón ha sido mal etiquetada.

4.4. Ajuste lineal.

Para poder realizar la correlación a lo largo del escalón, es necesario realizar un ajuste lineal. Esto se ha decidido hacer para encontrar la orientacion promedia del escalon y luego calcular la desviacion de esta orientacion.

Hasta ahora, se ha trabajado con píxeles, pero para realizar un ajuste lineal y después realizar la correlación, es necesario efectuar un cambio de píxeles a puntos reales. Para realizar dicho cambio, se ha llegado al acuerdo de que el punto real, estará localizado en el medio de un píxel.

Para poder representar un punto real a través de las coordenadas de un píxel, se ha realizado el siguiente cambio:

dRP = (distanciaRealImagen / distanciaPixelesImagen) * 10^9
Xreal = numeroPíxelesX + 0.5 * dRP
Yreal = numeroPíxelesY + 0.5 * dRP

numeroPíxelesX representa la coordenada de la columna, y numeroPixelesY la coordenada de la fila. dRP significa la distancia real de 1 píxel, que se calcula dividiendo la distancia real de la imagen por el número de píxeles. Esto último, se convierte de metros a nanómetros.

El cambio de coordenadas, de píxeles a reales, se hace cada vez que se elige una línea. Una vez tenemos los puntos reales, podemos realizar el ajuste lineal utilizando la función lineregress del módulo stats de la librería scipy. El ajuste lineal se hace 2 veces para conseguir los puntos con las coordenadas reales, y otro, con las coordenadas en píxeles. Se ha decidido hacer de esta forma, porque el cliente estaba interesado en 2 cosas: en mostrar el ajuste lineal y también en la ecuación de la recta del ajuste lineal en valores reales.

```
1 data.slopeReal, data.interceptReal, r_valueReal,p_valueReal, std_errReal = stats.linregress(data.
      XCoordReal, data.YCoordReal)
2
```

```
3 slopePixel, interceptPixel, r_valuePixel,p_valuePixel, std_errPixel = stats.linregress(data.
      XCoordPixel, data.YCoordPixel)
```

Una vez hecho el ajuste lineal, han sido necesarias 2 comprobaciones. En caso de que el escalón elegido solo tuviese 1 punto, sería imposible realizar el ajuste lineal. Para ello, mostramos un mensaje al usuario diciendo que el escalón seleccionado solo tiene 1 punto. En el caso de que los puntos del escalón seleccionado sean verticales, también sería imposible realizar el ajuste lineal. En este otro caso, también se informa al usuario de que los puntos del escalón seleccionado son verticales.

Si no se diese ninguno de los casos anteriores, se utiliza el algoritmo de Bresenham para la generación de la línea y poder mostrarla en la imagen. El algoritmo de Bresenham, fue creado para dibujar rectas en los dispositivos de gráficos rasterizados, como un monitor de ordenador, y determina qué píxeles se rellenan en función de la inclinación del ángulo de la recta a dibujar. En la figura 4.10, se puede apreciar un ejemplo.

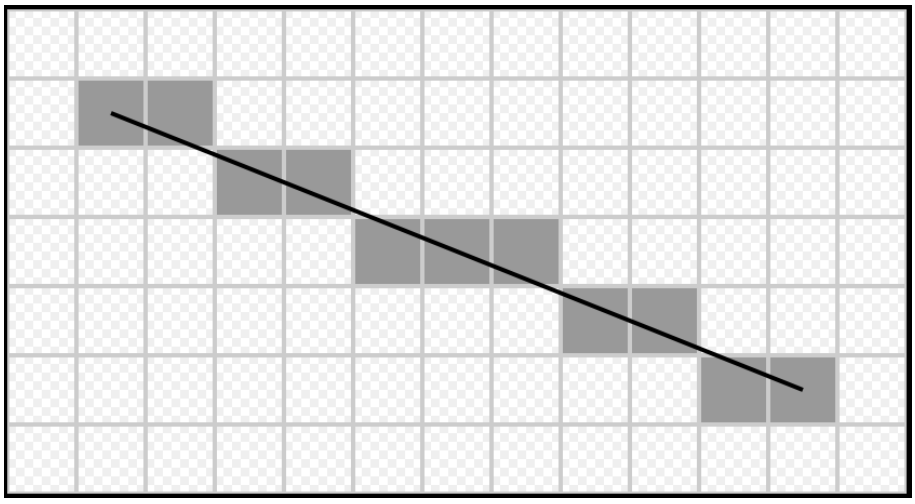

Figura 4.10: Una ilustración del resultado del algoritmo de Bresenham entre los puntos (0,0) y (10,4).

```
1  if len(data.XCoordPixel) == 1:
2     data.labelFittInfo.set_text("La traza solo tiene 1 punto.")
3  else:
4     iguales=True
5     primero = 0
6
7     for index in range(0,len(data.XCoordPixel)):
8        if data.XCoordPixel[0] != data.XCoordPixel[index]:
9           iguales=False
10    if iguales == False:
11       for index in range(0,tam[1]):
```

```
12          if fittedYPixel[index] >= 0 and fittedYPixel[index] <= 511:
13             if primero == 0:
14                firstCoord = (fittedYPixel[index],index)
15                primero = 1
16             lastCoord = (fittedYPixel[index],index)
17
18       points=bresenham((firstCoord[0],firstCoord[1]),(lastCoord[0],lastCoord[1]))
19
20       for index in range(0,len(points)):
21          ret2[points[index][1]][points[index][0]]=1
22
23       data.labelFittInfo.set_text("y=%sx+%s."%(data.slopeReal,data.interceptReal))
24    else:
25       data.labelFittInfo.set_text("Puntos verticales,imposible de ajustar")
```

A continuación, en las figuras 4.11, 4.12 y 4.13, se muestran los casos mencionados antes.

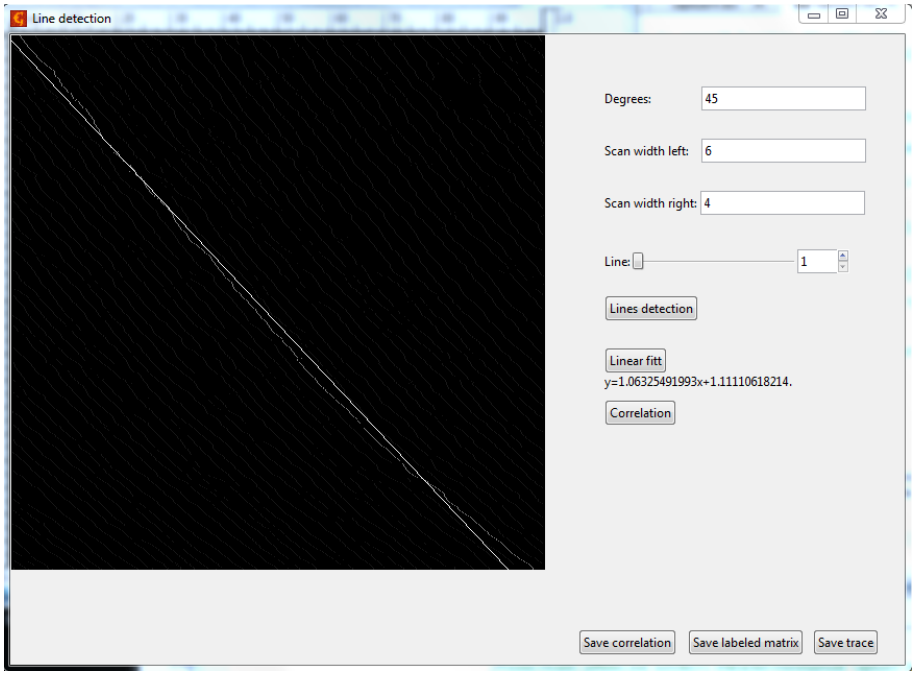

Figura 4.11: Se puede apreciar como se realiza el ajuste lineal sobre el primer escalón y se muestra la ecuación de la recta en nanómetros

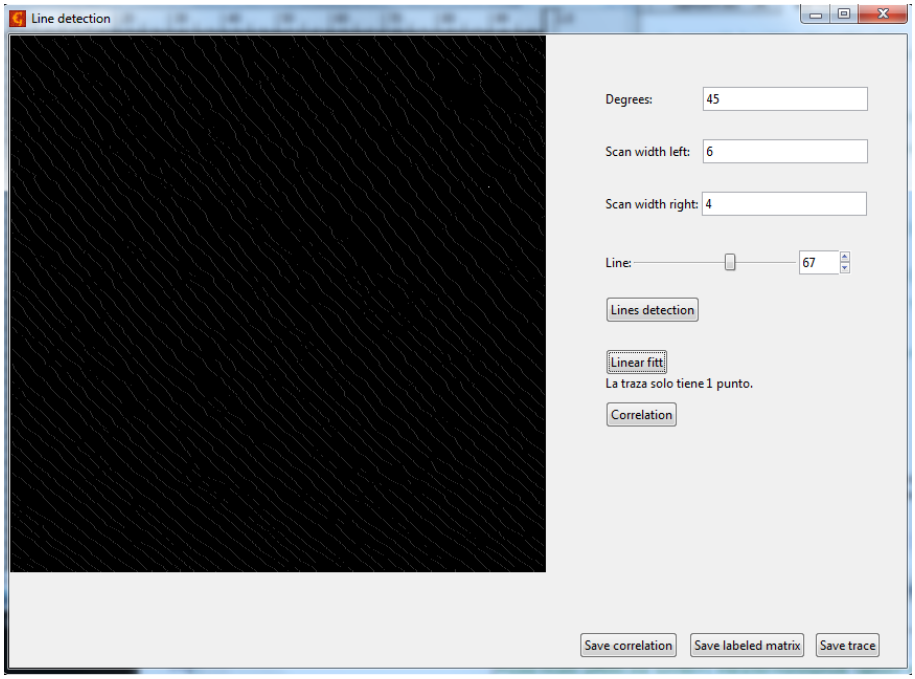

Figura 4.12: Se ha seleccionado el escalón número 67 y sólo tiene 1 punto. En este caso es imposible realizar el ajuste lineal y mostramos un mensaje.

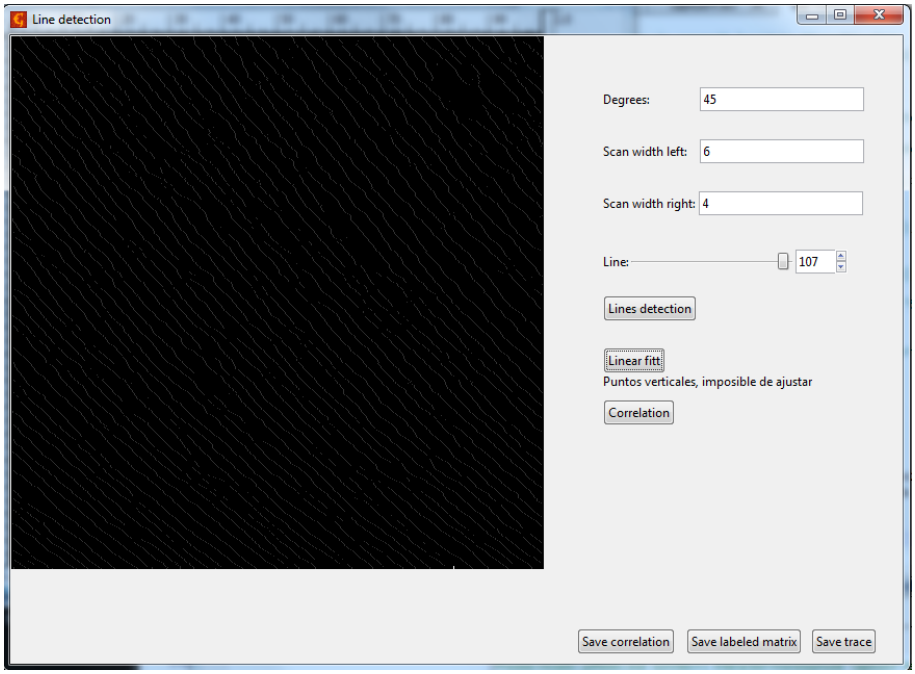

Figura 4.13: Se ha seleccionado el escalón número 107 y todos sus puntos son verticales. En este caso es imposible realizar el ajuste lineal y mostramos un mensaje.

4.5. Correlación a lo largo del escalón.

Por último, una vez se tiene el ajuste lineal, se hará una correlación con los puntos reales del escalón y se mostrará en una gráfica. Para realizar la correlación, se ha utilizado la siguiente función:

$$F(y) =< [x(y) - x(yo)]^2 >$$

Esta función, es la desviación promedia del escalón en la dirección perpendicular al ajuste lineal.

Para calcular la distancia en perpendicular de un punto a una recta, se ha utilizado la siguiente función:

$$d(A,D) = \frac{|a*x_A - y_A + b|}{\sqrt{a^2+1}}$$

"A" representa un punto y "D" una recta, donde la ecuación de "D" seria $y = a*x+b$ y la forma del punto "A" es igual a $A = (x_A, y_A)$. En la figura 4.14 se ha representado un punto "A" (un punto del escalón) y una recta "D" (ajuste lineal).

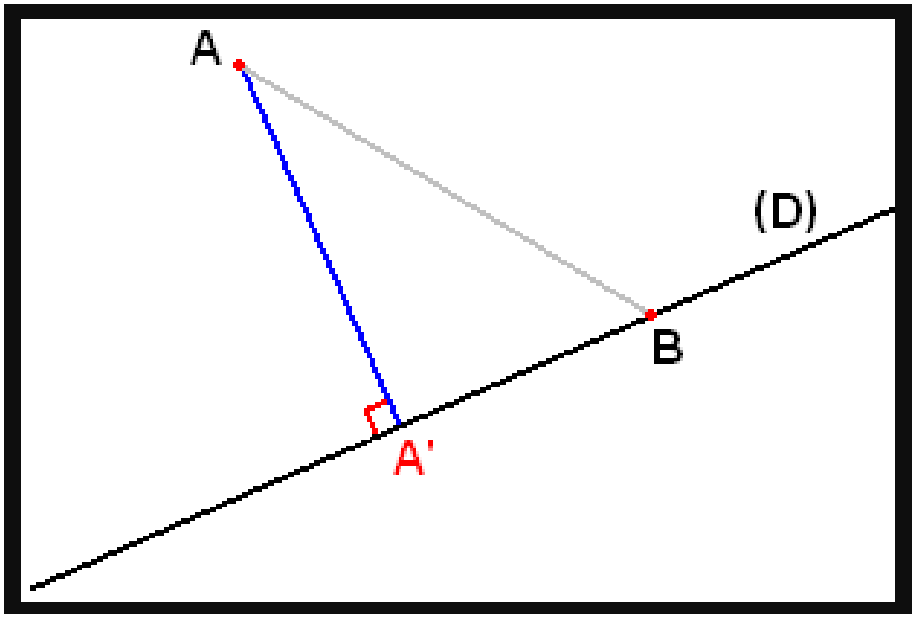

Figura 4.14: En geometría euclidiana, la distancia de un punto a una recta es la distancia más corta entre ese punto y un punto de una línea o recta.

A continuación, se muestra el código para realizar la correlación.

```
data.distanceCorr = []
dAD = 0
for index in range(0,len(data.XCoordReal)):
   dAD += np.absolute(data.slopeReal*data.XCoordReal[index]-data.YCoordReal[index]+data.
     interceptReal)/np.sqrt((data.slopeReal**2)+1)

   data.distanceCorr.append(dAD / (index+1))

plt.plot(data.distanceCorr)
plt.show()
```

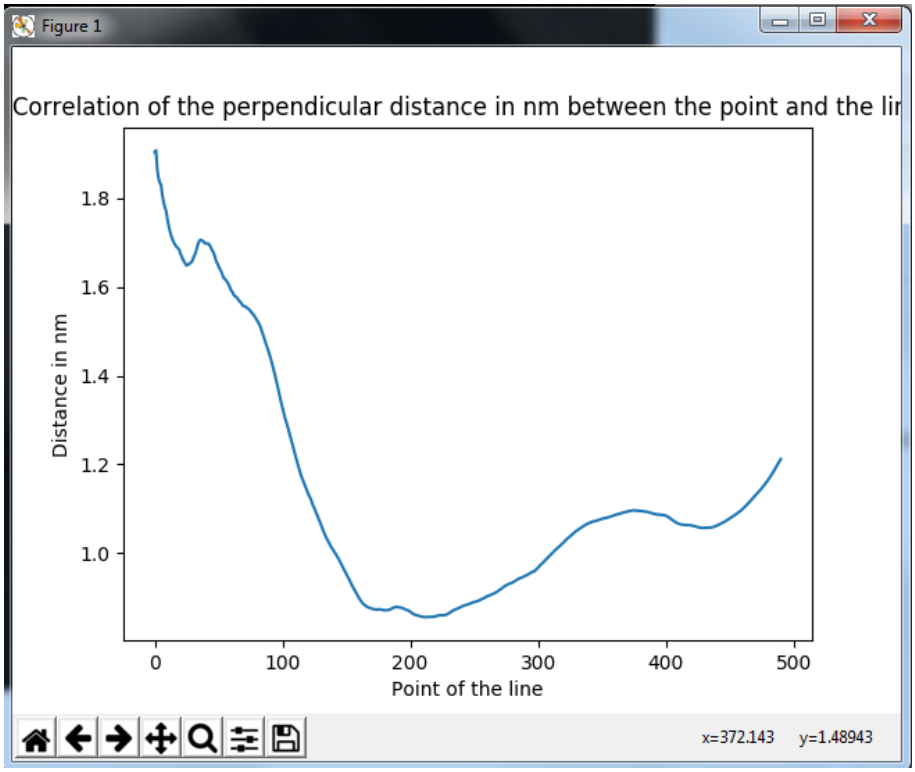

Figura 4.15: Se puede observar, punto a punto, la distancia perpendicular entre los puntos del escalón 1 y su ajuste lineal.

5. CAPÍTULO

GUI de los módulos.

5.1. Introducción

Para poder crear las interfaces gráficas de los 2 módulos mencionados en los 2 capítulos anteriores, se han utilizado las librerias de PyGTK, además de unos módulos de Gwyddion para poder interactuar con clases y objetos del mismo.

```
1 import  gtk, gobject, gwy, gwyutils
```

Un módulo con interaz gráfica de usuario contiene 3 partes: la definición del módulo, la construcción de la interfaz gráfica y el procesamiento de los datos.

La definición del módulo consta de 3 variables: plugin_menu, plugin_Desc y plugin_type.

- Plugin_menu: Es una variable de tipo string donde se escribe la ruta para acceder al módulo desde el programa Gwyddion.
- plugin_desc: Es una variable de tipo string donde se escribe una pequeña descripción de lo que realiza el módulo.
- plugin_type: Es una variable de tipo string donde se escribe a qué tipo pertenece el módulo (FILE, PROCESS, GRAPH, VOLUME y XYZ).

La construcción de la interfaz gráfica, es la parte donde añadimos la construcción de la ventana, con sus botones, cuadros para imagenes, ...

Por último, la parte del procesamiento de datos. Como su nombre indica, sería todo el procesamiento de datos que se va a realizar en el módulo.

Con el siguiente código, se muestra una pequeña plantilla con la estructura que deben de seguir los módulos que se deseén escribir en Python.

```
import gtk, gobject, gwy, gwyutils,...
#Definicion del modulo...
plugin_menu = "/Pygwy Examples/Ejemplo"
plugin_desc = "Breve descripcion del modulo"
plugin_type = "Tipo de modulo"

def run(data, mode):

#Construir y mostrar la GUI del modulo.

#Procesamiento de datos.
```

5.2. GUI del módulo para la detección de escalones atómicos e histograma de distancias.

Para crear este módulo, lo primero que se indica es la definición del módulo, además de importar todas las librerías que sean necesarias.

```
import gtk, gobject, gwy, gwyutils
import numpy as np
import matplotlib.pyplot as plt

# Define module info...
plugin_menu = "/Pygwy Examples/TWD"
plugin_desc = "TWD module"
plugin_type = "PROCESS"
```

Para construir la interfaz gráfica del módulo se han utilizado 2 funciones. La primera de ellas, windowCreate(data), será la responsable de crear la ventana de trabajo con sus dimensiones. La segunda, locate(data), incluye todos los widgets que se van a utilizar en este módulo y su localización en la ventana, como botones, cajas de texto, un cuadro para la imágen,... como se puede observar en la figura 5.1.

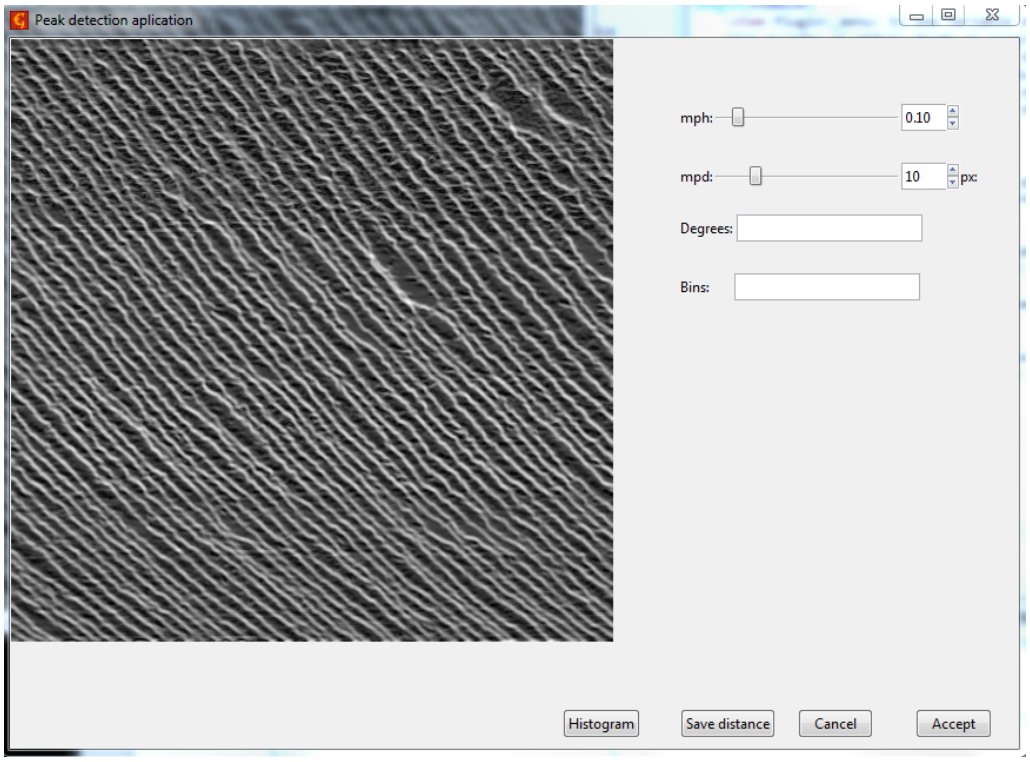

Figura 5.1: Interfaz gráfica del módulo para la detección de escalones atómicos e histograma de distancias.

La función run(), es la primera en ejecutarse cuando iniciamos el módulo, y tal y como hemos explicado antes, creamos la interfaz gráfica. En el siguiente código, se muestra una pequeña parte de como crear la interfaz gráfica.

```
def run(data, mode):
   data = windowCreate(data)
   locate(data)

def windowCreate(data):
   data.window = gtk.Window()
   data.window.set_title("Peak detection aplication")
   data.window.resize(858,604)
   data.window.show_all()
   return data

def locate(data):
   location = gtk.Fixed()
   data.window.add(location)

   ...
   data.scale_mph=gtk.HScale()
   ...
   data.spinn_mph=gtk.SpinButton()
```

```
20    ...
21    data.button_apply = gtk.Button(label="   Accept   ")
22    ...
```

El apartado para realizar el procesamiento de datos son el resto de funciones que se han incluido en el script, a las que se les llama mediante los widget de la ventana. Para poder llamar a estas funciones, se han realizado unas conexiones en las que un widget ejecutará una función cuando ocurra un evento en concreto, como puede ser un cambio de valor o clicar un botón. Para realizar dicha conexión, se utiliza la función connect con los parámetros del evento que tiene que ocurrir, la función a la que se desea llamar y el resto de argumentos que quiera pasar el desarrollador.

```
1  #widget.connect(evento, funcion_a_la_que_se_llama,argumento1, argumento2,...)
2  data.button_apply.connect("clicked", aceptar, data)
```

Este módulo, interactúa dinámicamente con la imagen para la detección de los escalones. Cuando cambiamos los parametros mph (minimun peak heigh) y mpd (minimun peak distance), mostramos al momento los cambios que sufre la imagen. Para ello, se ha utilizado la función aplicar(). En ella se recogen los nuevos valores mph y mpd, se utiliza la función data_field_data_as_array(field) para conseguir la matriz de la imagen original y se realiza un filtrado utilizando la función detect_peaks de **Marcos Duarte**, con la que se detectarán los picos. Una vez detectados los picos, estos se marcan en una nueva matriz para después volcarla en la imagen del módulo y poder ver los cambios. Para ellos, se utilizan las funciones data_changed() y data_field_set_data(field,array).

```
1  def aplicar(button,data):
2     mph = data.scale_mph.get_value()
3     mpd = data.scale_mpd.get_value()
4
5     data.array = gwyutils.data_field_data_as_array(
6     data.data_field)
7     data.array = np.transpose(data.array)
8
9     tam = data.array.shape
10    zerosa = np.zeros(tam)
11
12
13    for x in range (0,zerosa.shape[0]):
14       ind = detect_peaks(data.array[x], mph=float(mph),mpd=int(mpd), show=False)
15
16       for y in range (0, ind.size):
17          zerosa[x][ind[y]] = 1
18
```

```
19      data.array = zerosa
20      data.array = np.transpose(data.array)
21
22      data.another_field.data_changed()
23      gwyutils.data_field_set_data(data.another_field,data.array)
```

Uno de los intereses del cliente respecto a este módulo, ha sido la posibilidad de guardar los datos de las distancias entre escalones en un fichero de texto. Para ello, se ha utilizado la clase FileChooserDialog de gtk, con la que podemos mostrar una ventana y utilizarla para abrir ficheros o guardarlos. En el siguiente código, se puede ver un ejemplo de como hacerlo y el resultado de ello en la figura 5.2.

```
1   def datos(button,data):
2
3       dialogSave = gtk.FileChooserDialog("Save..", None,gtk.FILE_CHOOSER_ACTION_SAVE,
4              (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,gtk.STOCK_SAVE, gtk.RESPONSE_OK))
5
6       dialogSave.set_default_response(gtk.RESPONSE_OK)
7       response = dialogSave.run()
8
9       if response == gtk.RESPONSE_OK:
10          ...
11
12      elif response == gtk.RESPONSE_CANCEL:
13         print 'Closed, no files selected'
14      dialogSave.destroy()
```

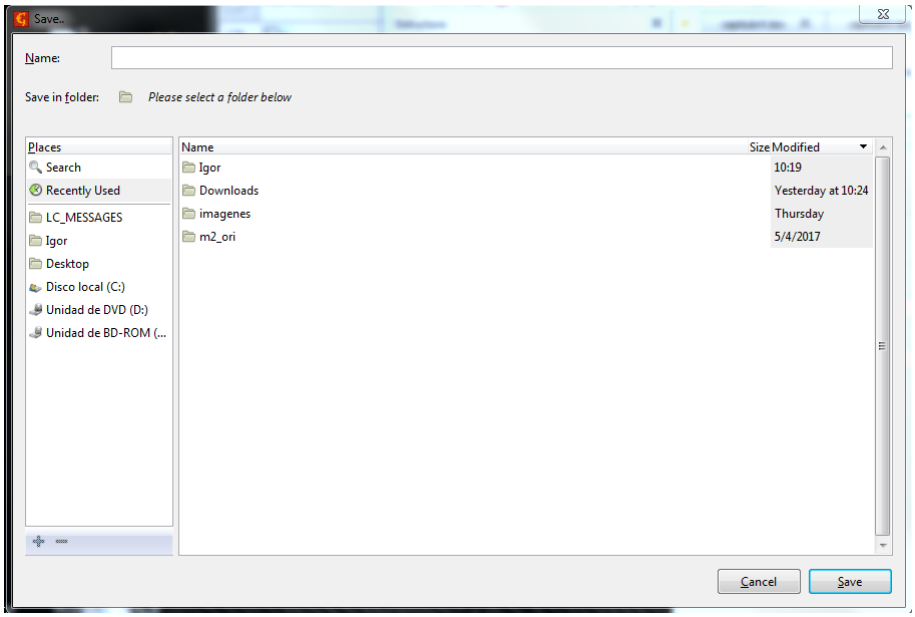

Figura 5.2: Interfaz gráfica de la clase FileChooserDialog para guardar datos en un fichero de texto.

5.3. GUI del módulo para el etiquetado y análisis estadístico de escalones.

Al igual que el módulo anterior, lo primero es crear la parte de su definición e importar las librerías necesarias.

```
1 import gtk, gobject, gwy, gwyutils
2 import numpy as np
3 from scipy import stats
4 import matplotlib.pyplot as plt
5
6 # Define module info...
7 plugin_menu = "/Pygwy Examples/Lines"
8 plugin_desc = "Lines module"
9 plugin_type = "PROCESS"
```

Para construir la interfaz gráfica de este módulo, se han utilizado las mismas 2 funciones que en el módulo anterior. La primera de ellas, windowCreate(data), será la responsable de crear la ventana de trabajo con sus dimensiones. La segunda, locate(data), incluye todos los widgets que se van a utilizar en este módulo y su localización en la ventana, como botones, cajas de texto, un cuadro para la imágen,... como se puede observar en la figura 5.3.

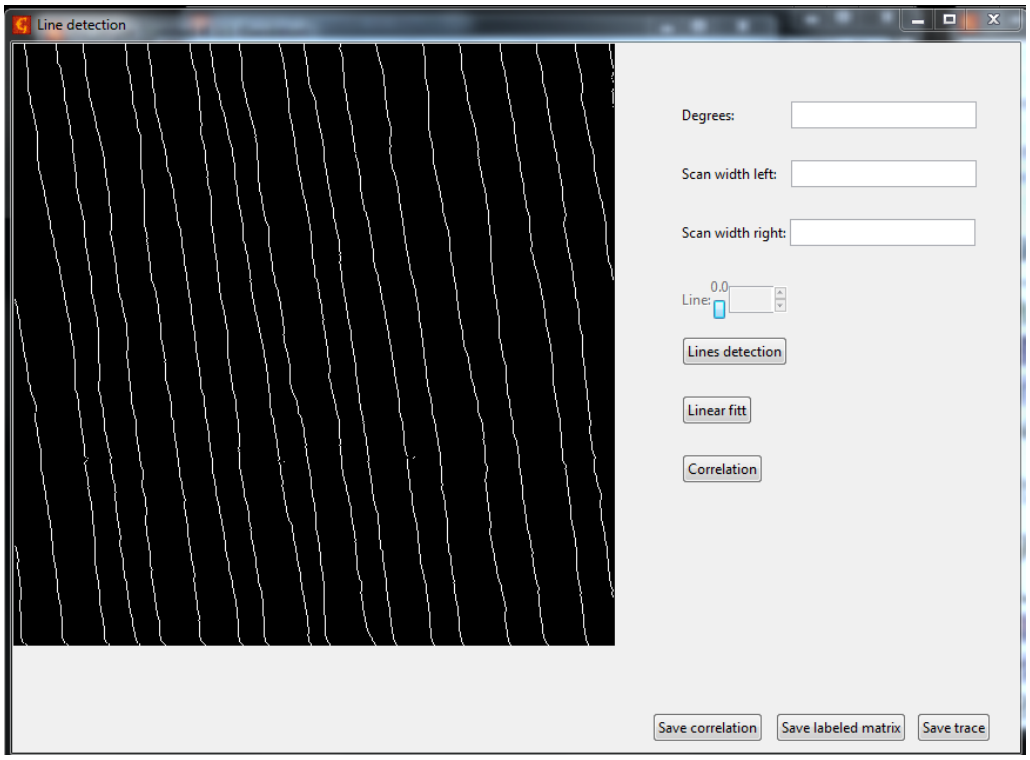

Figura 5.3: Interfaz gráfica del módulo para el etiquetado y análisis estadístico de escalones.

Este módulo también tiene la característica de ser dinámico, ya que una vez etiquetados todos los escalones, se permite al usuario elegir cada uno de ellos y visualizarlo en la imagen. En este módulo, a petición del cliente, se han añadido unos botones con los que guardar información específica, como la matriz etiquetada, las coordenadas del escalón seleccionado y los datos obtenidos mediante la correlación a lo largo de un escalón. Para ello, se ha utilizado la clase FileChooserDialog, con la que podemos abrir una ventana y nos permite indicar la ruta donde queremos guardar la información.

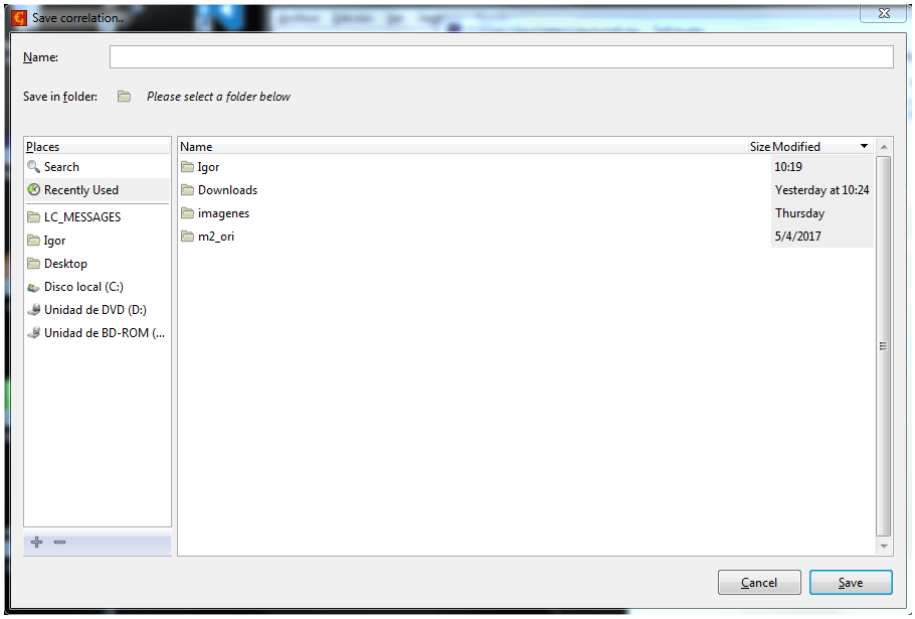

Figura 5.4: Interfaz gráfica de la clase FileChooserDialog para guardar los datos de la correlación en un fichero de texto.

6. CAPÍTULO

Gestión del proyecto.

En este capítulo, se presenta la gestión que se ha realizado para este proyecto, así como la estructura de descomposición del trabajo (EDT), la gestión del tiempo, la gestión de riesgos, las adquisiciones y las reuniones con los directores.

6.1. EDT.

Para terminar el proyecto exitosamente, se han definido dos apartados: el desarrollo del proyecto y su gestión. En la primera sección se engloba toda la investigación, la implementación de los módulos y la documentación, mientras que en la segunda sección, se trata toda la gestión de este proyecto.

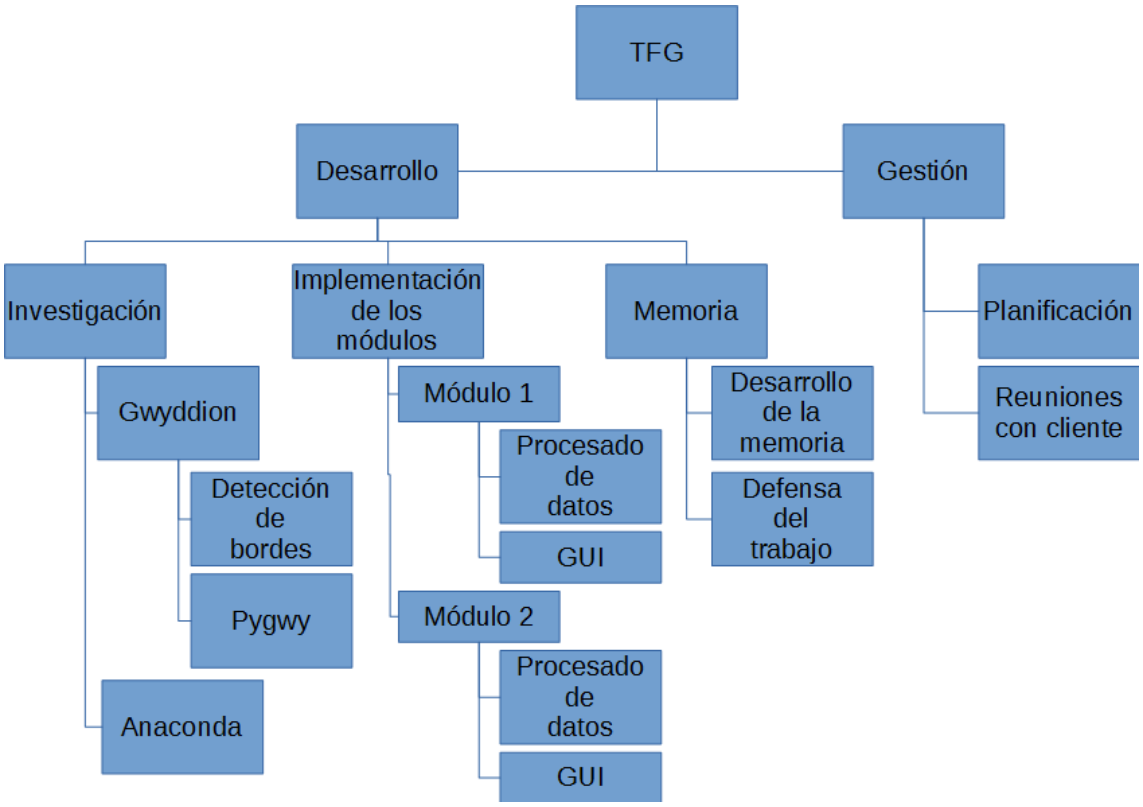

Figura 6.1: EDT.

6.2. Gestión del tiempo.

La ejecución de este proyecto comenzó el día 20 de Febrero. El planteamiento del proyecto para definir los objetivos ya estaban acordados, ya que antes de comenzarlo hubo unas reuniones para ver si se llegaba a un acuerdo. El final de este proyecto es el día 10 de Julio, que finaliza con su defensa. A continuación, en el cuadro 6.1, se puede ver la dedicación a cada tarea.

Tabla 6.1: Dedicación a cada tarea en horas.

Categoría	Tarea	Dedicación (h)
Investigación	Gwyddion	5
	Anaconda y liberías	1
	Python, PyGTK, Latex, Github,...	10
Implementación de los módulos	Módulo 1	105
	Módulo 2	90
Memoria	Desarrollo de la memoria	80
	Preparación de la defensa (estimación)	20
Gestión	Planificación	5
	Reuniones con los directores	8
Total:		**324**

6.3. Gestión de riesgos.

Al inicio del proyecto, se identificaron tres posibles casos que podían poner en riesgo el proyecto:

- El riesgo que mayor impacto podría tener en la ejecución de este proyecto, es no encontrar una solución a las posibles necesidades del cliente. En tal caso, el proyecto se tendría que abandonar, pero las probabilidades de que eso suceda son mínimas.

- El segundo riesgo, es no poder integrar los módulos dentro del programa Gwyddion. En tal caso, se crearán los programas con las mismas funcionalidades, pero se ejecutarán de forma independiente.

- El tercer y ultimo riesgo, es la perdida de todos los datos, tanto de los módulos como de la memoria. Para solucionar este problema, se utilizarán varios métodos de almacenamiento. El primero será Github, que además de guardar toda la información, es un sistema con control de versiones. En caso de perder la información o modificar los datos por unos erroneos, siempre se puede volver a una versión anterior. Otro método, será el almacenamiento en la nube de Google, que es accesible

desde cualquier lugar. Por último, siempre se tendrá una copia en local, tanto en el ordenador de trabajo como en el pendrive.

6.4. Adquisiciones.

Para la realización de este proyecto, se han adquirido dos funciones que han sido necesarias en el tratamiento de los datos:

- Función detect_peaks: Este algoritmo es capaz de detectar los picos según los parametros que más nos interesen. El autor se llama **Marcos Duarte** y ha sido adquirida de su cuenta de Github https://github.com/demotu/BMC.
- Función Bresenham: Este algoritmo es capaz de trazar una línea recta dados dos puntos. Se ha adquirido de la pagina web http://www.roguebasin.com/ index.php?title=Bresenham %27s_ que ofrece esta función en varios lenguajes, entre ellos Python.

6.5. Reuniones con los directores.

Durante la elaboración del proyecto, se ha mantenido contacto con los directores vía email y en reuniones. El seguimiento de las reuniones se puede ver a continuación:

- 8 de Marzo: Investigación y familiarización con los distintos algoritmos para la detección de bordes y el programa Gwyddion.
- 5 de Abril: Dudas sobre la estructura de la GUI de los módulos.
- 19 de Abril: Demostración del primer módulo.
- 25 de Mayo: Demostración del segundo módulo.
- 20 de Junio: Finalización del proyecto.

7. CAPÍTULO

Mejoras en los módulos.

A pesar de haber conseguido los objetivos que se habían planteado al inicio del proyecto, se podrían haber mejorado algunas cosas, así como automatizar la obtención de la inclinación de los escalones, utilizar un algoritmo para dibujar rectas con anti-aliasing y haber desarrollado los módulos utilizando el lenguaje de programación C.

7.1. Transformada de Fourier en la imagen para la obtención de la inclinación de los escalones

Un parámetro necesario para el análisis de los escalones en las imágenes, es el ángulo de inclinación que tienen. En este proyecto, se obtiene el ángulo utilizando una herramienta del programa Gwyddion, que permite rotar la imagen. Este paso, se podría automatizar si se utilizase la transformada de Fourier.

Estas pruebas están hechas en MATLAB, por lo que esta solución habría que modificarla y con la ayuda de las librerías científicas que se han utilizado, integrarla en los módulos. En el siguiente código, se mostrará como se ha obtenido el ángulo.

Lo primero es cargar la imagen, y convertir los datos a tipo double para poder trabajar con ellos.

```
1 % Cargar imagen
2 X = imread('m2_ori.par.bmp');
```

```
X = double(X(:,:,1))/256;
hfigX = figure;
imshow(X)
```

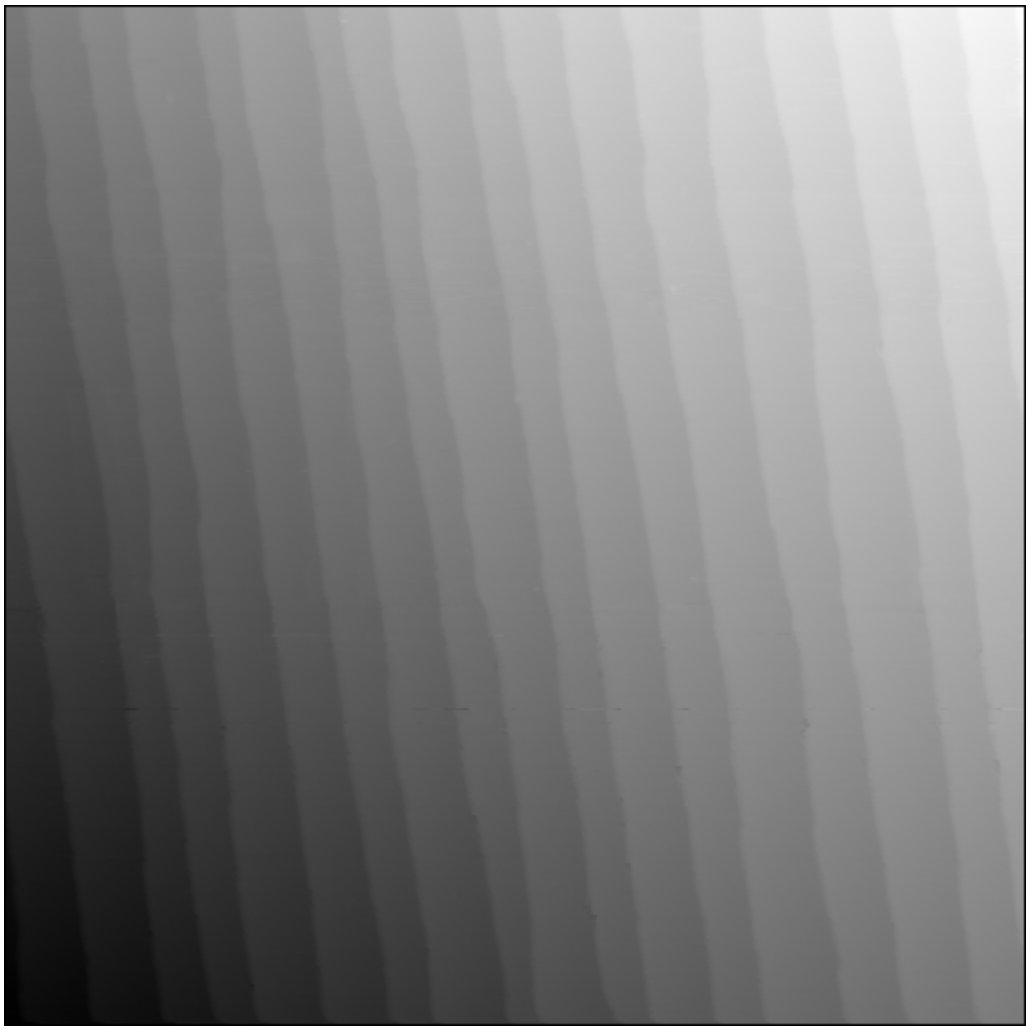

Figura 7.1: Imagen original.

Una vez se tiene la imagen, se le aplica una ventana de Hanning para eliminar los picos de los extremos y se obtiene el espectro de magnitud mediante la transformada de Fourier. Con esto, se crea una zona de búsqueda y se puede observar en la figura 7.2

```
% Espectro de magnitud
w = hanning(N);
X = (w*w').*X;
SM = abs(fft2(X));

% Zona de búsqueda
SMsearch = [SM((end-DMAX+1):end, 1:(1+DMAX)); SM(1:(1+DMAX), 1:(1+DMAX))];
```

```
ORG = [DMAX+1, 1];
SMsearch(ORG(1)+(-DMIN:DMIN), ORG(2)+(0:DMIN)) = 0;
figure, mesh(SMsearch/5), axis equal
```

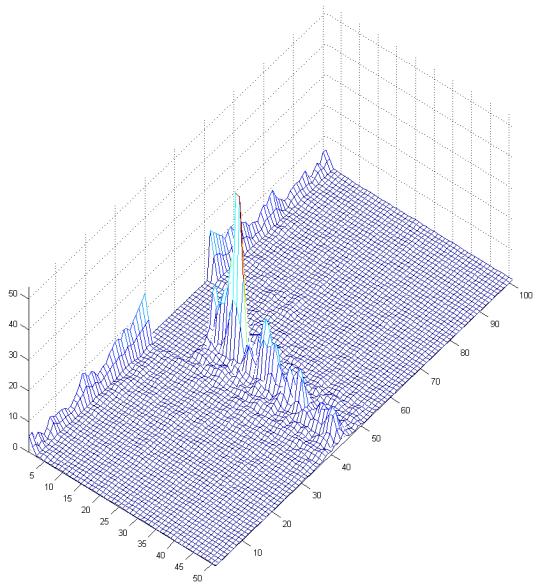

Figura 7.2: Representación del espectro de magnitud.

Ahora, se establece un umbral y se convierte a 0 todo lo que este por debajo, ya que no son datos que interesan. Esto se puede observar en la figura 7.3

```
% Recorte umbral 30%
SMsearch(SMsearch < 0.3*max(SMsearch(:))) = 0;
figure, mesh(double(SMsearch)/5), axis equal
```

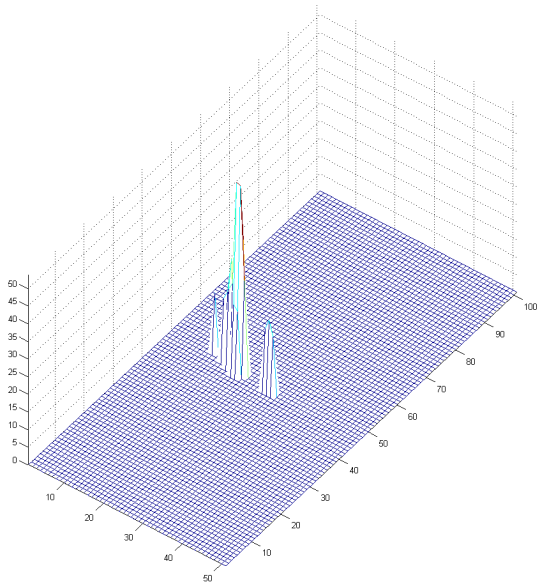

Figura 7.3: Se elimina todo lo que este por debajo del umbral, en este caso, un 30 %.

Una vez quedan los picos que nos interesan, se realiza un centro de gravedad entre ellos y se calcula el ángulo. En la figura 7.4, se muestra el punto y el ángulo de la zona de búsqueda.

```
% Centro de gravedad
K = (-DMAX:DMAX)'*ones(1, DMAX+1);
L = ones(2*DMAX+1,1)*(0:DMAX);
den = sum(sum(SMsearch));
pk = sum(sum(K.*SMsearch))/den;
pl = sum(sum(L.*SMsearch))/den;
angle = atan(pk/pl)*360/2/pi;
disp(['ángulo = ', num2str(angle), ' grados'])

% Dibujo punto y angulo resultante en zona de busqueda
figure, imshow(SMsearch, [])
hold on
plot(pl+1, DMAX+1+pk, 'ro')
plot([1, 2*pl+1], [DMAX+1, DMAX+1+2*pk], 'r')
hold off
```

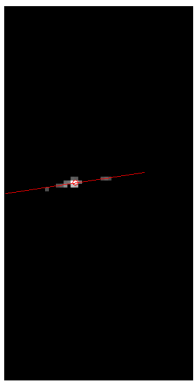

Figura 7.4: Se dibuja el punto y el ángulo resultante de la zona de búsqueda.

Por último, se superpone una línea con el ángulo que se ha obtenido y se comprueba si es correcto, figura 7.5.

```
% Dibujo linea superpuesta en imagen original
figure(hfigX)
hold on
plot([N/2 + pk/pl*N/2, N/2 - pk/pl*N/2], [0, N], 'r')
axis on
hold off
```

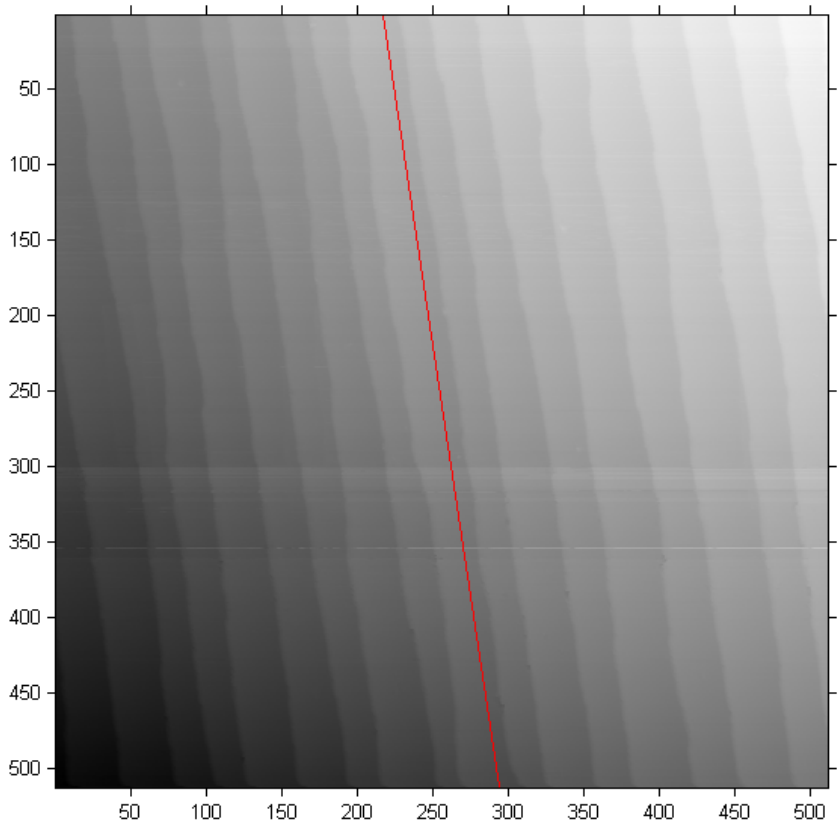

Figura 7.5: Se ha superpuesto una línea con el ángulo obtenido en la imagen, para comprobar si es paralela a los escalones de la imagen.

7.2. Algoritmo de Xiaolin Wu para dibujar rectas

El algoritmo que se ha utilizado en este proyecto para dibujar líneas rectas, ha sido el de Bresenham, ya que realiza operaciones muy sencillas y de bajo coste computacional, por lo que es muy rápido. En cambio, el algoritmo de Xiaolin Wu, es utilizado en la computación gráfica moderna, ya que soporta anti-aliasing, pero su computación no es tan sencilla, ya que necesita marcar un número de píxeles mayor con distintas intensidades.

El resultado de un algoritmo y otro, se puede ver en la siguiente figura 7.6. Si se fija en las líneas del fondo, se puede apreciar como en las líneas de la izquierda hay unos escalones, mientras que en las líneas de la derecha, se difuminan y parece una línea recta.

Figura 7.6: Lineas de la izquierda obtenidas con el algoritmo de Bresenham, líneas de la derecha obtenidas con el algoritmo de Xiaolin Wu.

Esto se debe a la manera de seleccionar los puntos que tiene cada algoritmo. El algoritmo de Bresenham, calcula la línea recta entre 2 coordenadas, y marca los puntos que están más cerca según la trayectoria de la línea. En cambio, el algoritmo de Xiaolin Wu, en cada paso, selecciona los 2 puntos que están más cerca de la trayectoria de la línea, y los colorea con diferente intensidad. En las figuras 7.7 y 7.8 se puede observar un ejemplo de cada algoritmo.

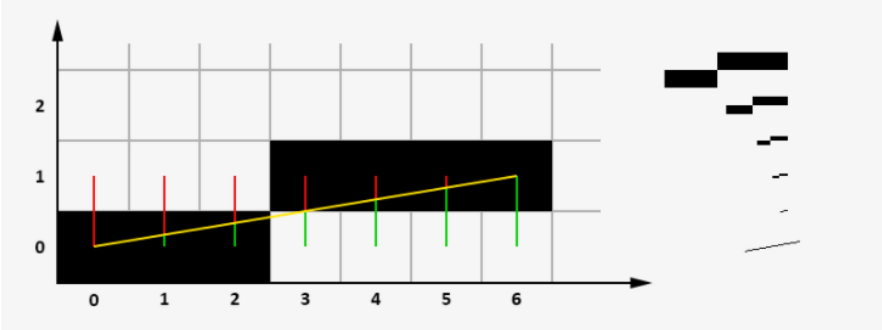

Figura 7.7: Ejemplo de los puntos que se marcan entre las coordenadas (0,0) y (1,6) utilizando el algoritmo de Bresenham.

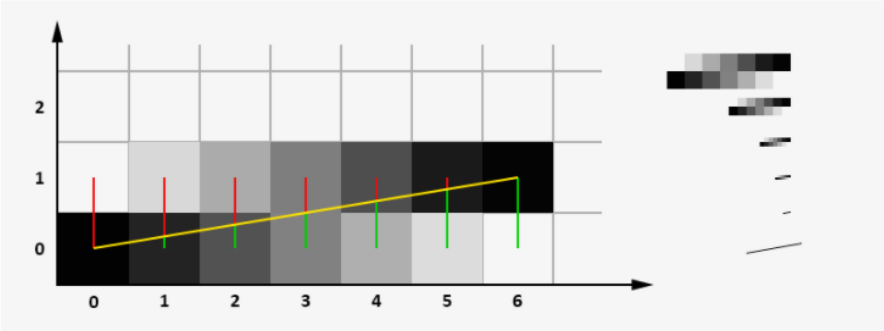

Figura 7.8: Ejemplo de los puntos que se marcan entre las coordenadas (0,0) y (1,6) utilizando el algoritmo de Xiaolin Wu.

7.3. Código de los módulos en C

A pesar de que existe la posibilidad de crear módulos en Python para el programa Gwyddion, todos los módulos que trae el programa vienen escritos en C. La mayor ventaja de escribir los módulos en C, es que existe la posibilidad de incluirlo dentro de los ficheros de compilación del programa si estuviesen de acuerdo sus principales desarrolladores, además de poder utilizar los módulos que ya trae como ejemplo, pero su complejidad a la hora de desarrollarlo y tener que compilar dichos ficheros cada vez que se realicen cambios, han sido los motivos por los que se ha rechazado este lenguaje.

8. CAPÍTULO

Conclusiones

En este capítulo se presentan las conclusiones del proyecto.

8.1. Conclusiones generales.

En el proyecto que se presenta, se han realizado 2 módulos para el programa de visualización y análisis de imágenes Gwyddion con éxito. El primer módulo, se encarga de realizar un filtrado para la detección de los escalones atómicos y calcular las distancias entre ellos. El segundo módulo, utiliza los resultados que se obtienen en el primer módulo, y enumera los distintos escalones para su posterior análisis estadístico.

Durante la realización del proyecto, se ha mantenido contacto con los directores, que pertenecen al Centro de Física de Materiales (CSIC-UPV/EHU) y están desarrollando un proyecto en el que es necesario la detección de escalones y su posterior análisis.

Al inicio del proyecto, el cliente estableció los requisitos, los cuales han sido los objetivos y han servido para definir el alcance de este proyecto. Al principio del proceso de la realización del proyecto, hubo un tiempo necesario para el aprendizaje del programa Gwyddion y sus distintas funcionalidades. Una vez familiarizado con el programa, se optó por la técnica para la detección de bordes que mejores resultados daba. Para proceder con el filtrado de estos bordes, se utilizó una función capaz de detectar los picos según nues-

tros criterios. Esta función se llama detect_peaks y su autor es **Marcos Duarte**. Durante la realización del segundo módulo, fue necesario crear un algoritmo capaz de detectar un escalón y etiquetarlo con un valor único, para después, poder realizar un estudio a lo largo de cada uno.

8.2. Conclusiones personales.

Desarrollar un proyecto de estas características no es comparable a ningún otro trabajo que haya podido elaborar en mi etapa como estudiante de la universidad o estudios anteriores. Es cierto que en todos estos años he aprendido muchas cosas y que he tenido que realizar muchos trabajos, pero la incógnita de no saber si se podría efectuar el proyecto, me causaban un mar de dudas al principio. Por otro lado, tener que desarrollar todo yo solo, era otro punto que me generaba inquietudes, pero gracias a lo aprendido en estos últimos 4 años con distintos profesores, compañeros de clase, la ayuda de mis directores y de San Google, esta ha sido una experiencia realmente positiva y gratificante.

La principal lección aprendida durante todo el proceso de realización de este proyecto, sería el lenguaje de programación utilizado, Python. Mis conocimientos sobre Python eran mínimos, pero con lo poco que sabía sobre ello, podía intuir que no sería difícil acostumbrarme a dicho lenguaje. Finalmente, puedo asegurar que es un lenguaje con una curva de aprendizaje no empinada, al menos para mi propósito.

El procesamiento de imágenes era algo que me llamaba mucho la atención, sobre todo desde que cursé la asignatura optativa de 4º Procesado Digital de Sonido e Imagen (PDSI). Aunque no he aplicado muchos de los conocimientos obtenidos en esta asignatura, se han añadido como posibles mejoras de los módulos.

El apartado más costoso de este proyecto, ha sido construir para los módulos unas interfaces gráficas para el usuario (GUI). Durante mi etapa como estudiante, en ninguna de las asignaturas he podido estudiar o trabajar en algún proyecto que estuviese relacionado con la construcción de unas GUIs.

Por último pero no menos importante, durante la realización de este proyecto, me he dado cuenta de que en la universidad, solo se obtiene una base para poder trabajar como graduado de ingeniería informática en cualquiera de sus ámbitos y que no debemos decir “a mi no me han preparado para esto”.

Anexos

A. ANEXO

Tutorial para el uso de los módulos

Una vez se ha ejecutado el programa Gwyddion, para abrir una imagen, pincharemos en File ⟶ Open A.1 y aparecerá la siguiente ventana A.2.

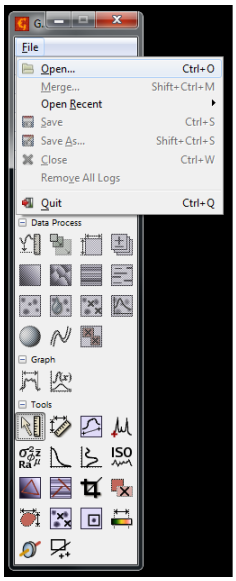

Figura A.1: Incio del programa.

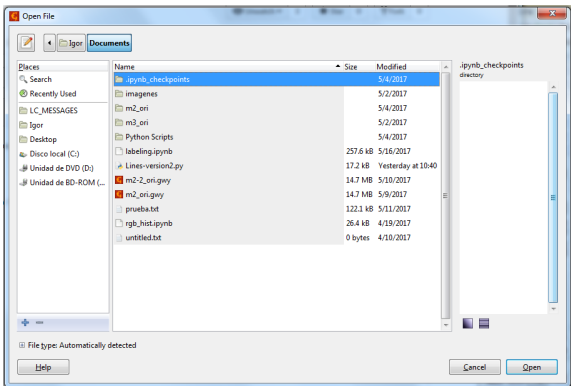

Figura A.2: Ventana para seleccionar imagenes..

Se selecciona la imagen con la que se quiere trabajar A.3 y se pincha en Open. Aparecerá en una nueva ventana con la imagen seleccionada A.4.

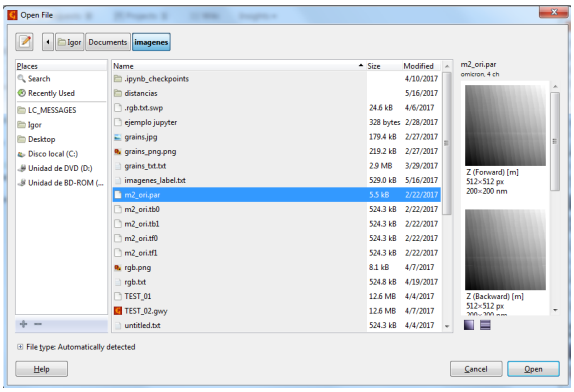

Figura A.3: Imagen seleccionada para abrir.

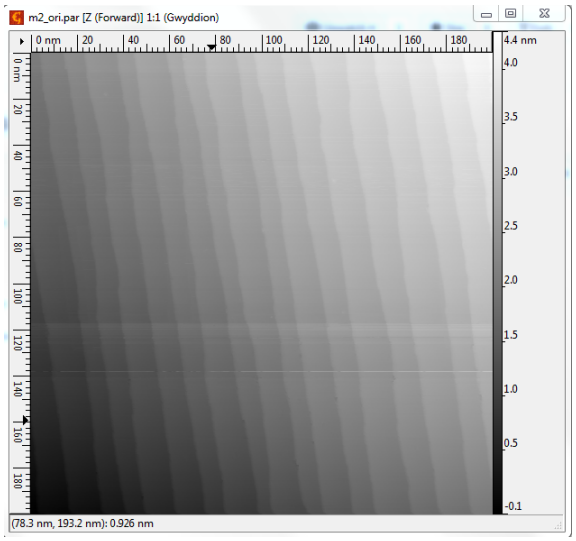

Figura A.4: Visualización de la imagen seleccionada.

Como primer tratamiento, se pueden eliminar los fallos que se observan en la imagen A.5.

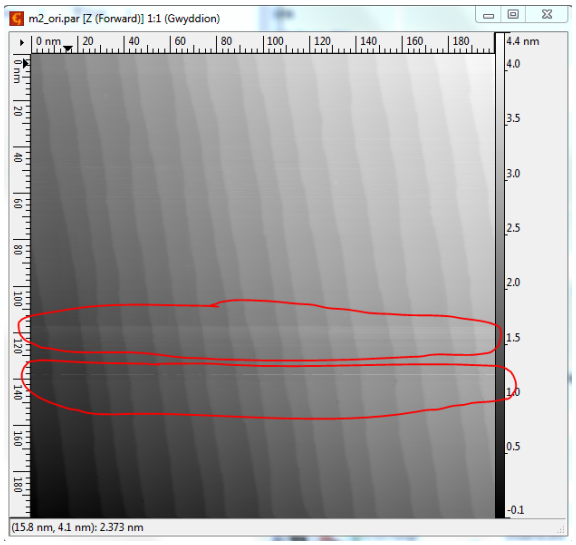

Figura A.5: Fallos en la imagen seleccionada.

Primero, se pincha en el boton Align rows using various methods A.6.

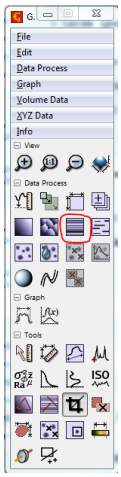

Figura A.6: Boton alignrows.

Después, se selecciona Median of differences y pinchamos en OK A.7.

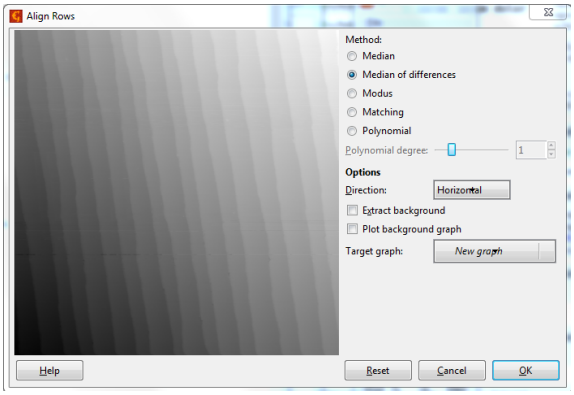

Figura A.7: Se selecciona median of differences.

Se puede apreciar como algunos fallos han sido eliminados. Ahora, se pincha en el boton Correct horizontal scars para eliminar un poco mas los fallos A.8.

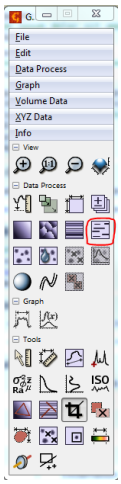

Figura A.8: Boton correct scars.

Se puede observar, que la imagen esta más limpia A.9.

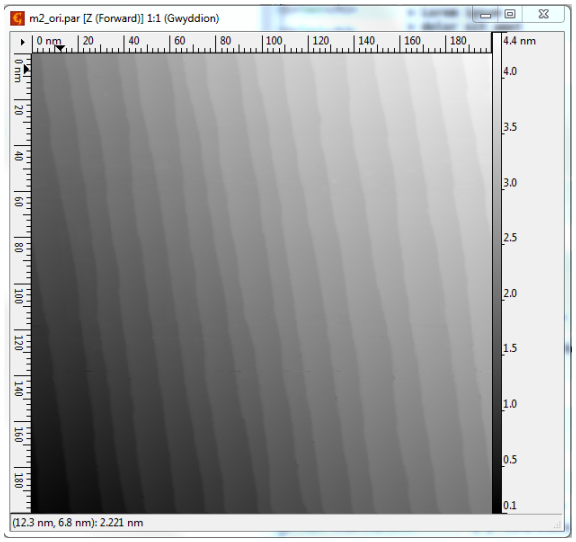

Figura A.9: Imagen seleccionada con fallos corregidos.

Para detectar los escalones de la imagen, primero se utilizará un algoritmo de detección de bordes. Tiene varios tipos de detección de bordes, en este caso se utilizá Inclination. Para ello, se selecciona en Data Process $\longrightarrow$ Presentation $\longrightarrow$ Edge Detection $\longrightarrow$ Inclination A.10.

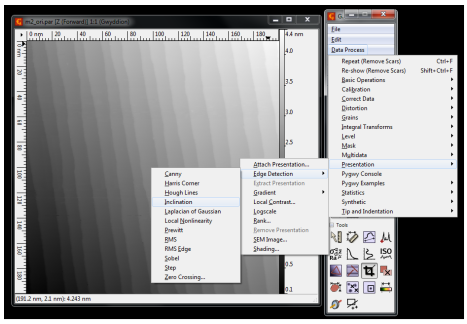

Figura A.10: Algoritmos para la detección de bordes.

Una vez se ha seleccionado la opción de inclination, la imagen quedará de esta forma A.11.

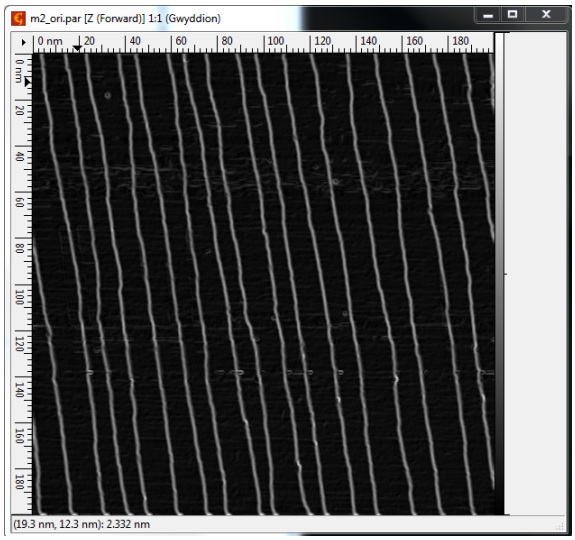

Figura A.11: Detectados los bordes de la imagen con el algoritmo de Inclination.

Es necesario que se extraiga la imagen para poder trabajar con esta nueva imagen, en caso contrario, se seguirá trabajando con la imagen anterior. Para ello, se selecciona en Data Process $\longrightarrow$ Presentation $\longrightarrow$ Extract Presentation A.12.

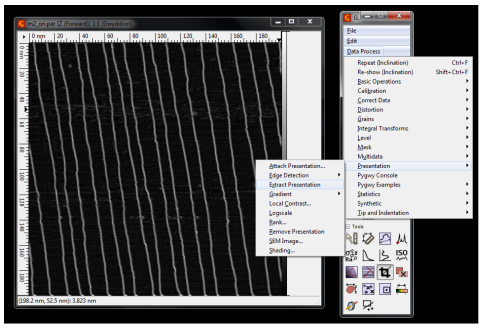

Figura A.12: Extraer la presentación.

Con esto, aparecerá una nueva imagen con la que se seguirá trabajando.

Ahora, se utilizará el módulo 1, para ello se selecciona en Data Process ⟶ Pygwy Examples ⟶ TWD A.13. Este módulo nos permite detectar los escalones y pasar a una matriz de 0 y 1. Esto nos abrirá una nueva ventana con la que podremos detectar los escalones A.14.

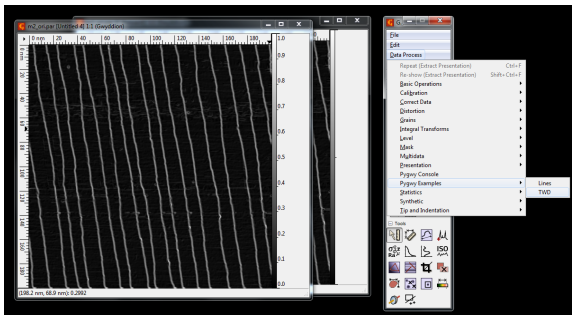

Figura A.13: Seleccionar módulos creados en Python.

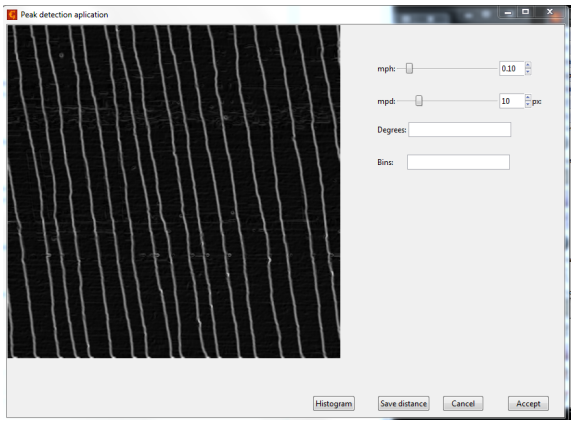

Figura A.14: GUI del módulo 1.

Será necesario ajustar los parametros mph (minimum peak height) y mpd (minimum peak distance). En este caso, se ajusta mph a 0,35 A.15. Si aceptamos, abrirá una nueva imagén con los escalones detectados.

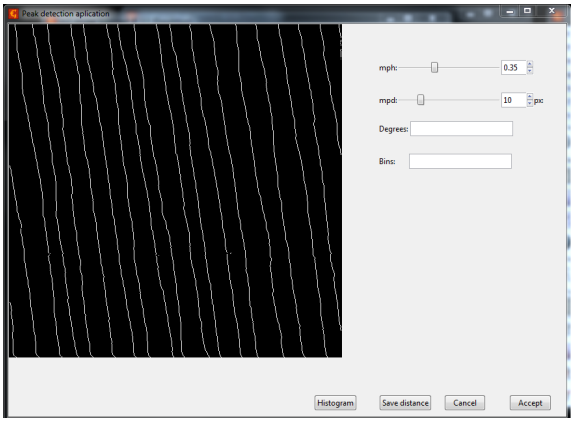

Figura A.15: Se le indica el valor de 0.35 al parametro mph.

Para guardar en un fichero las distancias entre escalones, será necesario indicar la inclinación de los escalones y pinchar en el boton Save distance. En este caso la inclinación es de 8 grados. Esto, abrirá una nueva ventana y se le indicará donde queremos guardar las distancias A.16 y A.17.

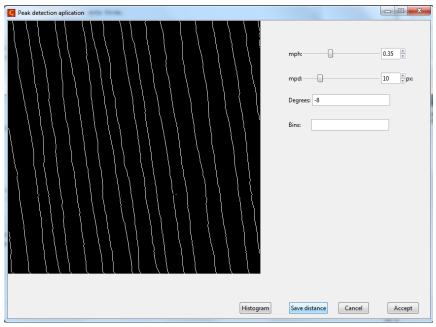

Figura A.16: Boton para almacenar las distancias entre escalones.

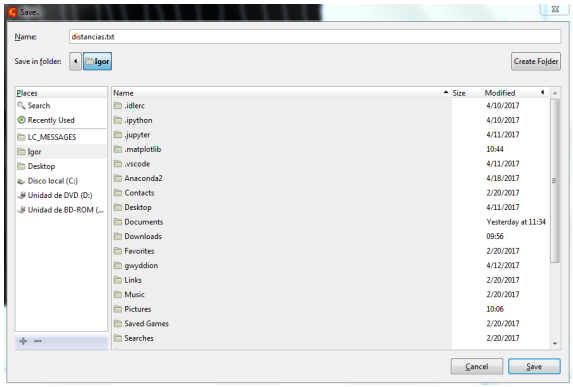

Figura A.17: Ventana para introducir el nombre del fichero donde se guardarán las distancias.

Para ver el histograma de las distancias, será necesario indicarle el bins (intervalo), en este ejemplo se utilizará un bins de 1 A.18 y A.19.

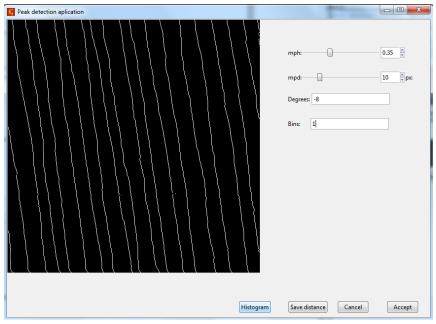

Figura A.18: Boton para mostrar el histograma de las distancias.

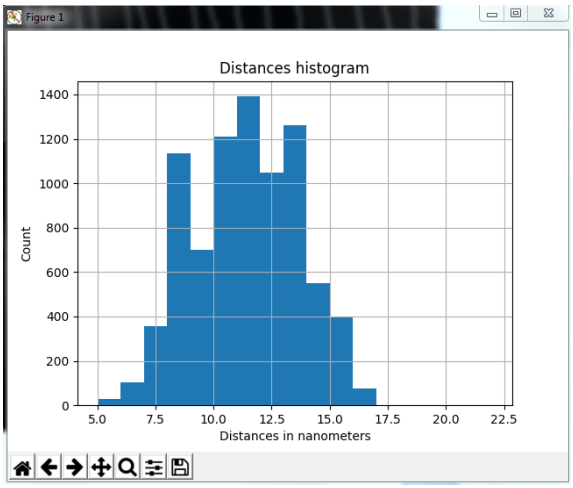

Figura A.19: Histograma con las distancias de la imagen A.9.

Con el módulo 2, se podrán etiquetar los escalones detectados y trabajar con cada uno de ellos independientemente de los demás. Para ello, se selecciona en Data Data Process ⟶ Pygwy Examples ⟶ Lines A.20. Esto abre nueva ventana A.21.

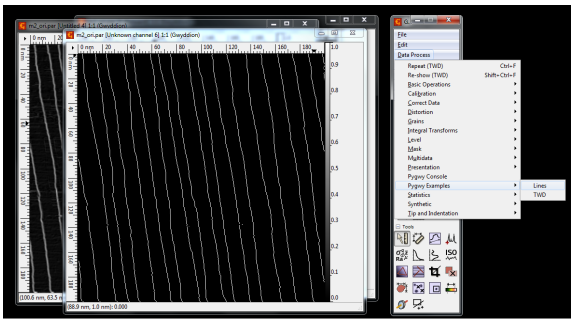

Figura A.20: Seleccionar módulos creados en Python..

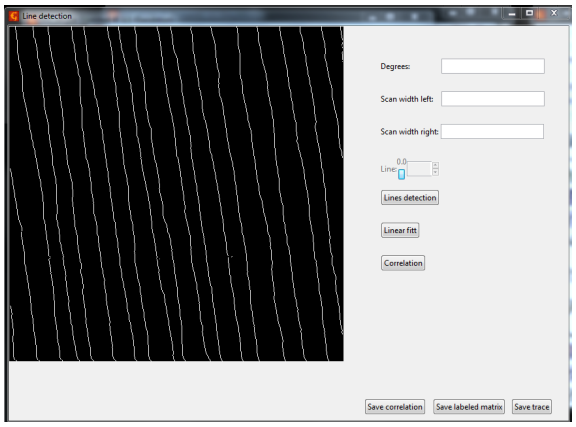

Figura A.21: GUI del módulo 2.

Para poder etiquetar los escalones de la imágen y seleccionar uno de ellos, primero se indicará la inclinación de los escalones y que margen se quiere utilizar para detectar los escalones. En este caso, se ha utilizado la inclinación de 8, y un margen de 7 a la izquierda y 7 a la derecha A.22. Pinchamos en Lines detection, y se puede observar en la figura A.23, que la línea 1 ha sido seleccionada.

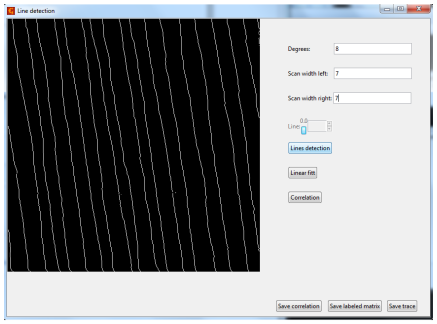

Figura A.22: Boton para seleccionar lineas independientemente de las demás.

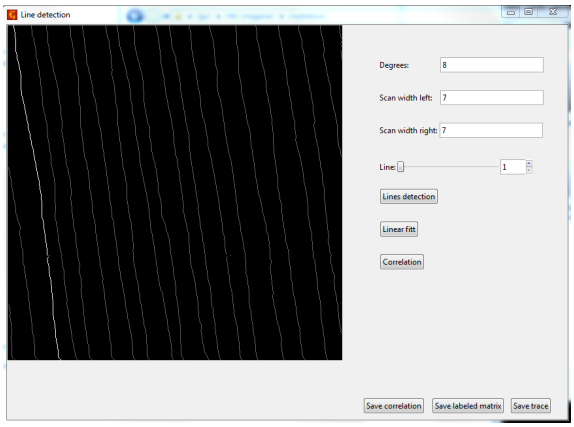

Figura A.23: Primer escalón seleccionado.

Una vez detectados los escalones, se puede seleccionar la línea que se quiera y realizar varias operaciones. Se puede ejecutar un ajuste lineal, representar la correlación del escalón con el ajuste lineal, guardar las coordenadas del elemento seleccionado y guardar la matriz con los escalones etiquetados como los datos de la correlación.

Bibliografía

[1] Gwyddion user guide. URL:
http://gwyddion.net/download/user-guide/gwyddion-user-guide-en.pdf

[2] PyGTK. URL:
http://www.pygtk.org/pygtk2reference/

[3] Función detect_peaks. URL:
http://nbviewer.jupyter.org/github/demotu/BMC/blob/master/notebooks/DetectPeaks.ipynb

[4] Función algoritmo Bresenham. URL:
http://www.roguebasin.com/index.php?title=Bresenham%27s_Line_Algorithm#Python

[5] Algoritmo de Bresenham VS Xiaolin Wu's
https://unionassets.com/blog/algorithm-brezenhema-and-wu-s-line-299

[6] Distancia de un punto a una recta. URL:
https://es.wikipedia.org/wiki/Distancia_de_un_punto_a_una_recta

[7] Stack Overflow. URL:
https://es.stackoverflow.com/

[8] Google. URL:
https://www.google.es/

[9] Wikipedia. URL:
https://es.wikipedia.org/wiki/Wikipedia:Portada